Θέμα Διπλωματικής Εργασίας:

«Κατανεμημένο Σύστημα Προσομοίωσης Προβλημάτων Πολλαπλών-Χωρίων/Πολλαπλών-Φυσικών Μοντέλων»

Συγγραφέας Διπλωματικής
Μπούσια Αλεξάνδρα

Επιβλέποντες Καθηγητές
Τσομπανοπούλου Παναγιώτα
Βάβαλης Μανόλης

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών
Τηλεπικοινωνιών και Δικτύων
Πανεπιστήμιο Θεσσαλίας, Ελλάδα

Οκτώβριος 2008

…αφιερωμένη σε όσους είναι πάντα δίπλα μου…

... Ευχαριστίες ...

Θέλω να ευχαριστήσω εξαιρετικά την επιβλέπουσα καθηγήτρια μου κ. Παναγιώτα Τσομπανοπούλου για την όλη υποστήριξη που μου προσέφερε κατά την διάρκεια της φοίτησης μου, αλλά και κατά την εκπόνηση της διπλωματικής μου εργασίας. Οι ιδέες της, τα σχόλια και οι προτάσεις και οι προτροπές της όπως επίσης και οι συνεχείς συναντήσεις μας, αποτέλεσαν την κινητήρια δύναμη για την εκπλήρωση της εργασίας μου...

Να ευχαριστήσω επίσης και τον καθηγητή κ. Μανόλη Βάβαλη που ήταν επίσης πάντα δίπλα μου και πάντα πρόθυμος να απαντήσει στις ερωτήσεις και να με συμβουλέψει στις απορίες μου, ενώ παράλληλα μου έδινε κουράγιο στις τελευταίες μέρες της εργασίας μου...

Ακόμη, θα ήθελα να δώσω τις θερμότερες ευχαριστίες μου σε όλους τους καθηγητές του τμήματος Μηχανικών Ηλεκτρονικών Υπολογιστών Τηλεπικοινωνιών και Δικτύων που με βοήθησαν καθ' όλη την διάρκεια των σπουδών μου και μου προσέφεραν γνώσεις και με έκαναν να αγαπήσω το μελλοντικό μου επάγγελμα. Ήταν μεγάλη μου τιμή που τους συνάντησα...

Σε αυτό το σημείο να ευχαριστήσω τους γονείς μου και τον αδελφό μου. Ιδιαίτερα τον πατέρα στον οποίο χρωστάω εν μέρει την επιλογή της σχολής από την οποία θα αποφοιτήσω. Δεν θα ξεχάσω ποτέ τις συμβουλές της μητέρας μου που πάντα με στηρίζει και με προτρέπει να μην αγχώνομαι. Ένα μεγάλο ευχαριστώ και στον αδελφό μου, Βασίλη, για τα αστεία του, το χιούμορ του και την αγάπη του...

Τέλος, ένα μεγάλο ευχαριστώ στον Κώστα που υπήρξε πάντα δίπλα μου και με στήριξε σε κάθε αγωνία και σε κάθε δυσκολία. Η ηρεμία του χαρακτήρα του μου χάρισε την ισορροπία που μου ήταν αναγκαία στις δύσκολες στιγμές. Είναι ένας άνθρωπος που με στηρίζει με ένα τρόπο μοναδικό και ξεχωριστό...

Πίνακας Περιεχομένων

Κεφάλαιο 1: Εισαγωγή

Η προσομοίωση και η μοντελοποίηση των πολύπλοκων συστημάτων απαιτεί την χρήση πολλών συνιστωσών, καθώς τα φυσικά συστήματα αποτελούνται από συστατικά διαφορετικής φυσικής σύστασης, οι παράλληλες υπολογιστικές προϋποθέτουν ανεξάρτητα συστατικά και τα υπάρχοντα λογισμικά επιλύουν μόνο απλά γεωμετρικά συστήματα. Νέα λογισμικά και μοντέλα προσομοίωσης έχουν προταθεί στις δύο προηγούμενες δεκαετίες, ενώ έχουν μελετηθεί μέθοδοι για *interface relaxation* σε προβλήματα πολλαπλών χωρίων/ πολλαπλών φυσικών μοντέλων. Η παρούσα διπλωματική εργασία έχει ως στόχο να παρουσιάσει ένα νέο περιβάλλον προσομοίωσης για την επίλυση μερικών διαφορικών εξισώσεων. Παρέχουμε μια γραφική διεπαφή (Graphical User Interface - GUI), όπου ο χρήστης θα μπορεί να ζωγραφίσει ένα δυσδιάστατο χωρίο και να ορίσει τις συνθήκες για τα εξωτερικά και τα εσωτερικά όρια. Επίσης, ο χρήστης θα έχει την δυνατότητα να ορίσει την διαφορική εξίσωση και τελικά να την επιλύει και να βλέπει την λύση.

Υπάρχουν τρεις υπολογιστικές προσεγγίσεις για την προσομοίωση μεγάλων επιστημονικών προβλημάτων. Η *πρώτη* τεχνική βασίζεται στην διακριτοποίηση του γεωμετρικού χωρίου χρησιμοποιώντας πλέγματα (grids or meshes) καθώς και μαθηματικά μοντέλα για τα σημεία των διεπαφών. Η *δεύτερη* και πιο παλιά μέθοδος είναι η διάσπαση Schwarz. Σε αυτή την τεχνική διαχωρίζεται το γεωμετρικό χωρίο σε μικρότερα που όμως επικαλύπτονται. Τα μαθηματικά μοντέλα λύνουν το κάθε επιμέρους χωρίο και μετά εφαρμόζεται η μέθοδος Schwarz για να υπολογιστεί η συνολική λύση. Η επικάλυψη των χωρίων δημιουργεί μεγάλη πολυπλοκότητα ακόμη και σε απλά προβλήματα. Η *τρίτη* και πιο νέα μέθοδος είναι η μέθοδος του interface relaxation. Σύμφωνα με αυτή την μέθοδο το χωρίο χωρίζεται σε υποχωρία, όπου το καθένα έχει ξεχωριστό μαθηματικό μοντέλο. Στα σημεία των διεπαφών εφαρμόζονται συγκεκριμένες συνθήκες, που βασίζονται και ικανοποιούν φυσικά φαινόμενα (π.χ. θερμοκρασία, διατήρηση ορμής). Αφού προσεγγιστούν οι τοπικές λύσεις με βάση τις τιμές στις διεπαφές, υπολογίζεται τελικά η ολική λύση.

Οι παραπάνω τρεις μέθοδοι έχουν κοινό στόχο να χειρίζονται πολύπλοκα και διαφορετικά φυσικά μοντέλα. Διαφέρουν όμως στην πολυπλοκότητα. Η μέθοδος Schwarz με επικάλυψη δημιουργεί ισχυρή σύζευξη μεταξύ των γειτονικών υποχωρίων. Οι τεχνικές χωρίς επικάλυψη περιορίζονται σε ένα και μοναδικό μαθηματικό μοντέλο για τα γειτονικά υποχωρία. Η τεχνική των interface relaxation δεν δημιουργούν συνθήκες σύζευξης και δίνουν τα καλύτερα αποτελέσματα. Η μέθοδος των interface relaxation είναι μια επαναληπτική διαδικασία που ακολουθεί τα παρακάτω βήματα για να επιλύσει το συνολικό πρόβλημα των μερικών διαφορικών εξισώσεων:

1) Αρχικά, αρχικοποιούνται τις τιμές στις διεπαφές των υποχωρίων.
2) Επιλύονται οι διαφορικές εξισώσεις για κάθε υποχωρίο βασιζόμενες στις εκτιμόμενες τιμές των διεπαφών.
3) Βελτιώνονται οι τιμές πάνω στις διεπαφές χρησιμοποιώντας διάφορες μεθόδους.
4) Επιστρέφει στο βήμα 2 εάν δεν ικανοποιούνται οι προϋποθέσεις που θέτονται όσο αφορά το σφάλμα και το πλήθος των επαναλήψεων.

Κεφάλαιο 2: Μέθοδοι Interface Relaxation για Μερικές Διαφορικές Εξισώσεις

Οι μέθοδοι για την διακριτοποίηση χωρίων που έχουν προταθεί, έχουν σχεδιαστεί για την επίλυση ελλειπτικών διαφορικών εξισώσεων. Όπως αναφέραμε και παραπάνω, οι μέθοδοι μπορούν να κατηγοριοποιηθούν σε μεθόδους με επικάλυψη και σε τεχνικές με μη επικάλυψη. Και οι δύο κατηγορίες χρησιμοποιούνται για συστήματα μεγάλης κλίμακας. Οι μέθοδοι με επικάλυψη, όπως είναι η μέθοδος Schwarz ήταν ιδιαίτερα σημαντική, αλλά σύμφωνα με νέες έρευνες έχει αποδειχθεί και πειραματικά πως οι μέθοδοι χωρίς επικάλυψη είναι πιο αποτελεσματικές και δεν απαλλάσσουν τον χρήστη από την πολυπλοκότητα της υλοποίησης που απαιτούσαν οι μέθοδοι με επικάλυψη.

Οι μέθοδοι του interface relaxation μας πηγαίνουν ένα βήμα πιο μπροστά από τις μεθόδους διάσπασης του χωρίου χωρίς επικάλυψη. Στην προσπάθεια τους να προσομοιώσουν τον πραγματικό κόσμο, διασπούν το πολύπλοκο πρόβλημα των μερικών διαφορικών εξισώσεων, το οποίο ορίζεται σε ένα μεγάλο και πολύπλοκο χωρίο, σε μικρότερα υποχωρία. Με αυτό τον τρόπο έχουμε το Πολλαπλό-Σύστημα Διαφορικών Εξισώσεων/ Πολλαπλών-Χωρίων μοντέλο, το οποίο χρησιμοποιεί παραμέτρους εξομάλυνσης για τις διεπαφές.

Σε αυτό το σημείο θα μελετήσουμε τις μεθόδους του interface relaxation για την επίλυση των ελλειπτικών διαφορικών εξισώσεων. Οι μέθοδοι αυτές βασίζονται στην διαμέριση του χωρίου σε μη επικαλυπτόμενα υποτμήματα και στην χρήση κατάλληλων συνθηκών στα εσωτερικά κομμάτια. Στην συνέχεια, χρησιμοποιώντας κάποιες αρχικές υποθέσεις για τις διεπαφές, λύνουμε τις εξισώσεις για τα επιμέρους υποχωρία. Αν οι λύσεις που βρούμε δεν ικανοποιούν τις συμβάσεις που έχουμε θέσει για τις διεπαφές, τότε επαναλαμβάνουμε την διαδικασία για να πάρουμε καλύτερες τιμές. Τα βήματα αυτά επαναλαμβάνονται μέχρι να συγκλίνουμε στην λύση του προβλήματος.

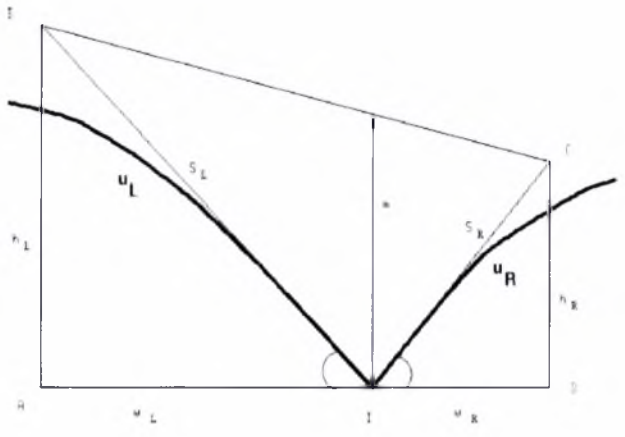
Οι πιο γνωστές μέθοδοι για interface relaxation είναι οι παρακάτω:

AVE : Μια απλή μέθοδος που υπολογίζει κατά μέσο όρο την λύση και τις παραγώγους πάνω στις διεπαφές.
GEO : Μια μέθοδος βασισμένη στην απλή γεωμετρική διάσπαση.
NEW : Μια τεχνική που βασίζεται στην μέθοδο του Newton για την «διόρθωση» των τιμών των διεπαφών.
ROB : Ένας αλγόριθμος που χρησιμοποιεί τις Robin διεπαφές για την εξομάλυνση της λύσης.
SCO : Μια τεχνική που βασίζεται στο συμπλήρωμα του Schur.
SHO : Μια μέθοδος που βασίζεται στην ιδέα του shooting για την επίλυση των Μερικών Διαφορικών Εξισώσεων.
SPO : Μια μέθοδος που προήλθε από την μαθηματική συνάρτηση των Steklov-Poincare, η οποία περιλαμβάνει την εναλλαγή τύπων στις διεπαφές.

Στην παρούσα διπλωματική εργασία, μία μέθοδος αναπτύσσεται παρακάτω και είναι η μέθοδος GEO:

- Μέθοδος GEO

Η μέθοδος GEO υπολογίζει την νέα λύση για κάθε υποχωρίο λύνοντας ένα Dirichlet πρόβλημα και θεωρείται μέθοδος ενός βήματος. Οι τιμές στις διεπαφές υπολογίζονται αν στις παλιές τιμές προσθέσουμε ένα γεωμετρικό συνδυασμό των παραγώγων των γειτονικών υποχωρίων. Στο παρακάτω σχήμα (Σχήμα 1) υποθέτουμε πως οι u_L και u_R είναι οι λύσεις του προβλήματος των Μερικών Διαφορικών Εξισώσεων στο αριστερό και το δεξί υποχωρίο αντίστοιχα στην διεπαφή I. Για να πάρουμε την ολική λύση καταλαβαίνουμε εύκολα από την γεωμετρία πως το m είναι η παράμετρος που πρέπει να προσθέσουμε στα u_L και u_R.



Σχήμα 1.

Αλγοριθμικά η μέθοδος GEO φαίνεται παρακάτω:

for k=1, 2, . . .

$$ui^{(k+1)} = ui^{(k)} - \frac{w_L w_R}{w_L + w_R}\left(\frac{\vartheta u_L^{(k)}}{\vartheta x} - \frac{\vartheta u_R^{(k)}}{\vartheta x}\right) \text{ σε κάθε διεπαφή,}$$

$$u^{(k+1)} = solvepde(ui^{(k+1)}) \text{ σε κάθε υποχωρίο.}$$

End

Κεφάλαιο 3: Υπάρχον λογισμικό για την επίλυση Μερικών Διαφορικών Εξισώσεων

Το Matlab παρέχει ένα ισχυρό και ευέλικτο περιβάλλον για την μελέτη και την επίλυση μερικών διαφορικών εξισώσεων. Χρησιμοποιώντας απλά την εντολή *pdetool* ανοίγει ένα γραφικό περιβάλλον το οποίο μας παρέχει τις παρακάτω δυνατότητες:

1) Ορισμός του προβλήματος των μερικών διαφορικών εξισώσεων, δηλαδή σχεδίαση του χωρίου, ορισμός των εξωτερικών συνθηκών και των συντελεστών της εξίσωσης.
2) Αριθμητική επίλυση του προβλήματος με την δημιουργία πλέγματος, την διακριτοποίηση των εξισώσεων και τελικά την προσέγγιση της λύσης.
3) Οπτικοποίηση των αποτελεσμάτων.

Το εργαλείο *pdetool* είναι σχεδιασμένο τόσο για αρχάριους όσο και για προχωρημένους χρήστες. Η βασική εξίσωση που επιλύει είναι η ελλειπτική διαφορική εξίσωση, που δίνεται από τον τύπο:

$$-\nabla \cdot (c\nabla u) + au = f$$

Η επίλυση λοιπόν των διαφορικών εξισώσεων είναι πολύ απλή με την χρήση του εργαλείου *pdetool* και περιλαμβάνει τα ακόλουθα βήματα:

Βήμα 1^{ο}: Ορισμός του προβλήματος χρησιμοποιώντας το *pdetool*

Ο πιο απλός τρόπος για τον ορισμό ενός προβλήματος διαφορικών εξισώσεων είναι με την χρήση του γραφικού περιβάλλοντος που παρέχει το *pdetool*. Στο Draw mode, μπορούμε να δημιουργήσουμε το χωρίο Ω, δηλαδή την γεωμετρία του προβλήματος, χρησιμοποιώντας ένα σετ αντικειμένων (τετράγωνο, κύκλος, έλλειψη και πολύφωνο) που παρέχει. Συνθέτοντας αυτά τα αντικείμενα δημιουργούμε την φόρμουλα της γεωμετρίας που επιθυμούμε. Στο Boundary mode, καθορίζουμε τις συνθήκες των εξωτερικών τμημάτων. Μπορούμε να έχουμε διαφορετικούς τύπους συνθηκών όπως για παράδειγμα Dirichlet, Neumann και μεικτούς. Στο *pdetool* οι διεπαφές μεταξύ των χωρίων δεν λαμβάνονται υπόψη. Στο PDE mode, μπορούμε να καθορίσουμε τον τύπο της εξίσωσης και να θέσουμε τιμές στους συντελεστές *a*, *c*, *f* και *d*. Μπορούμε να καθορίσουμε διαφορετική εξίσωση για κάθε υποχωρίο.

Βήμα 2^{ο}: Επίλυση του προβλήματος χρησιμοποιώντας το *pdetool*

Τα περισσότερα προβλήματα μπορούν να επιλυθούν με την χρήση του γραφικού περιβάλλοντος *pdetool*. Στο Mesh mode, παράγουμε τα πλέγματα, ενώ παράλληλα μπορούμε να καθορίσουμε και διάφορες παραμέτρους. Στο Solve mode, μπορούμε να υλοποιήσουμε και να ελέγξουμε μη γραμμικούς αλλά και προσαρμόσιμους τρόπους επίλυσης για τις ελλειπτικές εξισώσεις. Για παραβολικά και υπερβολικά προβλήματα, μπορούμε να καθορίσουμε τις αρχικές τιμές και τον αριθμό των επαναλήψεων. Αφού λύσουμε το πρόβλημα μπορούμε να επιστρέψουμε στο Mesh mode για να βελτιώσουμε το πλέγμα και να λύσουμε και πάλι το πρόβλημα.

Βήμα 3^{ο}: Οπτικοποίηση των αποτελεσμάτων

Από το γραφικό περιβάλλον μπορούμε να χρησιμοποιήσουμε το Plot mode, όπου έχουμε πολλές διαφορετικές δυνατότητες παρουσίασης της λύσης. Μπορούμε να χρησιμοποιήσουμε ξεχωριστές εικόνες.

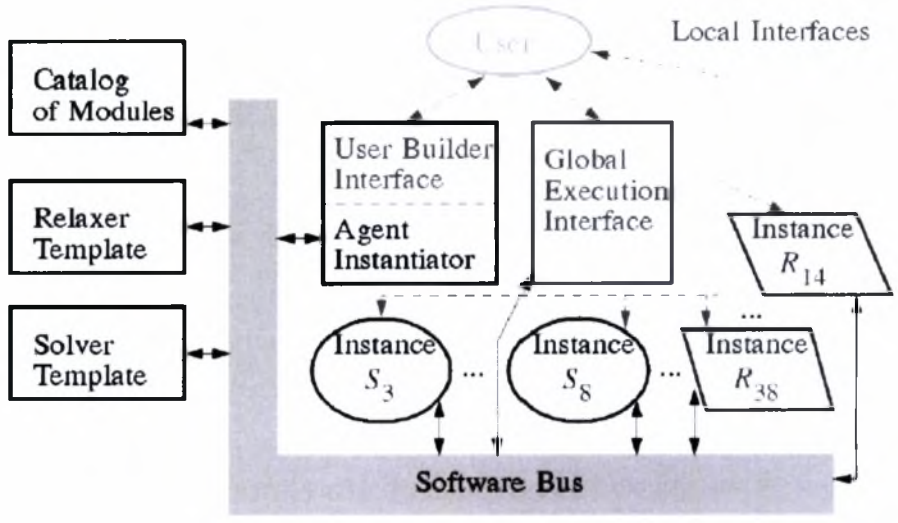
Κεφάλαιο 4: Υπάρχον λογισμικό για την μέθοδο του Interface Relaxation

Η μεθοδολογία του Interface Relaxation για την επίλυση των μερικών διαφορικών εξισώσεων είναι σχετικά νέα και επομένως δεν υπάρχουν πολλές υλοποιήσεις που να την καλύπτουν. Μία απλή υλοποίηση παρουσιάστηκε το 1991, βασιζότανε στο TCP/IP και έγινε για την συνεργασία μεταξύ συναδέλφων. Δεν χρησιμοποιούσε τμήματα λογισμικού και ανέπτυσσε μόνο μία μέθοδο, την AVE. Μια ακόμη απλοϊκή υλοποίηση παρουσιάστηκε και διαφέρει από την πρώτη καθώς χρησιμοποιούσε KQML μηνύματα και πράκτορες (ELLPACK PSE).

- Υλοποίηση Networked Agents for Scientific Computing

Ο Drashansky παρουσίασε μία προσομοίωση για πολύπλοκα μοντέλα, τα οποία περιλαμβάνουν περίπλοκες συνθήκες και γεωμετρία. Το Περιβάλλον Επίλυσης Προβλημάτων Πολλαπλών Καθηκόντων (MPSE) είναι κατάλληλο για αυτό τον σκοπό καθώς μπορεί να εφαρμοστεί για πολλές εφαρμογές και πρακτικά προβλήματα, επιτρέπει την επαναχρησιμοποίηση λογισμικού, έχει χαμηλό κόστος και υψηλή ποιότητα.

Πιο συγκεκριμένα, για να λύσουμε ένα πρόβλημα με βάση το MPSE αν 1) το πρόβλημα αποτελείται από πιο απλά, συνδεδεμένα και πιθανώς ετερογενή τμήματα, 2) κάθε τμήμα μπορεί να μοντελοποιηθεί με ένα απλό μαθηματικό μοντέλο και 3) οι διεπαφές μεταξύ των διαφόρων τμημάτων μπορούν να προσαρμοστούν με την χρήση κατάλληλων συνθηκών στις διεπαφές και επικοινωνία μεταξύ των γειτονικών χωρίων. Στο περιβάλλον προσομοίωσης που παρουσιάζει, υπάρχουν δύο βασικοί τύποι πρακτόρων- οι solvers και οι mediators. Ο κάθε solver πράκτορας υπολογίζει την τοπική λύση του προβλήματος στο υποχωρίο. Ο solver αντιμετωπίζεται σαν «μαύρο κουτί» από τους άλλους πράκτορες και αλληλεπιδρά με τους υπόλοιπους χρησιμοποιώντας μία ειδική γλώσσα επικοινωνίας. Από την άλλη πλευρά, οι mediator πράκτορες είναι υπεύθυνοι για να ρυθμίζουν τις διεπαφές μεταξύ δύο γειτονικών υποχωρίων. Κάθε mediator αναλαμβάνει μία διεπαφή, ενώ όσο πιο πολύπλοκη είναι η διεπαφή μπορεί να υπάρχουν περισσότεροι πράκτορες για την ρύθμιση των συνθηκών της. Οι mediators αναλαμβάνουν την ανταλλαγή δεδομένων μεταξύ των solvers που υπολογίζουν τις λύσεις στα γειτονικά υποχωρία. Οι μέθοδοι που μπορούν να εφαρμοστούν στην διεπαφή ποικίλουν ανάλογα με την φύση του προβλήματος. Τελικά, το δίκτυο μεταξύ των solvers και των mediators πρακτόρων οδηγεί στην εύρεση της συνολικής λύσης του προβλήματος. Η αρχιτεκτονική του MPSE από την πλευρά του χρήστη φαίνεται στην εικόνα 1. Από την πλευρά του ο χρήστης καλείται να ορίσει τα υποχωρία, να καθορίσει τα μοντέλα και τις διαφορικές εξισώσεις για κάθε υποχωρίο και να ρυθμίσει τις φυσικές συνθήκες στις διεπαφές. Ένα παράδειγμα ενός MPSE συστήματος είναι το PYTHIA.



Εικόνα 1. Αρχιτεκτονική του MPSE.

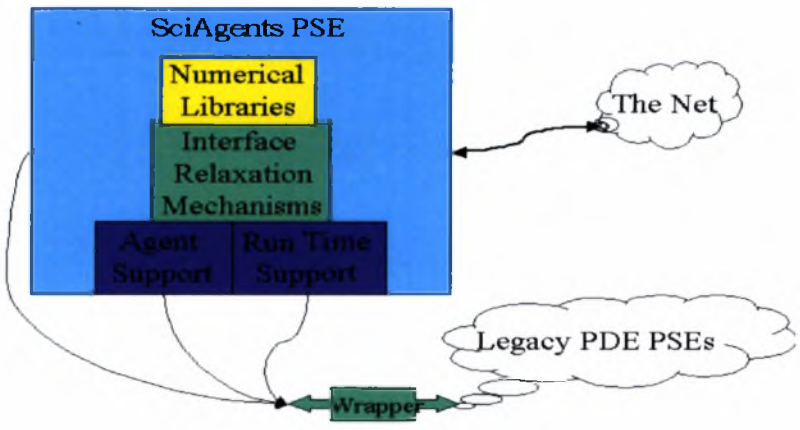
- Υλοποίηση RELAX

Ο McFaddin παρουσίασε το εργαλείο RELAX, ένα πειραματικό σύστημα για την επίλυση πολύπλοκων διαφορικών εξισώσεων. Η προσέγγιση αυτή βασίζεται σε αντικείμενα-τμημάτων. Σε μία μονάδα περιγράφεται η γεωμετρία του προβλήματος και σε άλλο τμήμα καταγράφονται οι συνθήκες και τα υπόλοιπα δεδομένα του προβλήματος και τέλος η λύση παρουσιάζεται σε μία ξεχωριστή διαδικασία. Οι μονάδες αυτές συνδυάζονται κατάλληλα σε ένα αντικειμενοστραφές περιβάλλον που έχει τις παρακάτω δυνατότητες:

- Γεωμετρία: Ο χρήστης δημιουργεί χωρία, τα οποία συνθέτουν το συνολικό πρόβλημα, χρησιμοποιώντας βασικά σχήματα.
- ΜΔΕ: Ορίζονται διαφορετικές διαφορικές εξισώσεις και κατάλληλες συνοριακές συνθήκες. Σε κάθε υποχωρίο.
- Μέθοδοι επίλυσης: Ο χρήστης μπορεί να καθορίσει διαφορετικές μεθόδους για την επίλυση του προβλήματος καθώς επίσης μπορεί αν τροποποιήσει και τις παραμέτρους τους.
- Διεπαφές: Οι διεπαφές που ορίζονται από τα υποχωρία θεωρούνται ως αυτόνομα αντικείμενα στο εργαλείο RELAX.
- Μέθοδοι Interface Relaxation: Παρέχονται ένα πλήθος διαφορετικών μεθόδων τις οποίες μπορεί να επιλέξει ο χρήστης.
- Σχέδιο δράσης: Καθορίζεται για να συγχρονιστούν όλες οι διεργασίες.

- Υλοποίηση SciAgent

Ένα πιο ολοκληρωμένο εργαλείο για την επίλυση γραμμικών και μη γραμμικών προβλημάτων με μερικές διαφορικές εξισώσεις είναι το σύστημα SciAgent σύστημα, το οποίο υλοποιήθηκε με την χρήση των γλωσσών C και Java. Το σύστημα αυτό εκμεταλλεύεται τον έμφυτο παραλληλισμό των μεθόδων Interface Relaxation. Τα τμήματα από τα οποία αποτελείται ένα SciAgent σύστημα φαίνονται στην εικόνα 3.



Εικόνα 2. Τμήματα του SciAgent.

Πιο συγκεκριμένα, για την επίλυση ενός προβλήματος με το εργαλείο SciAgent, χρησιμοποιείται ένα δίκτυο με τοπικούς solvers διαφορικών εξισώσεων και relaxers διεπαφών.

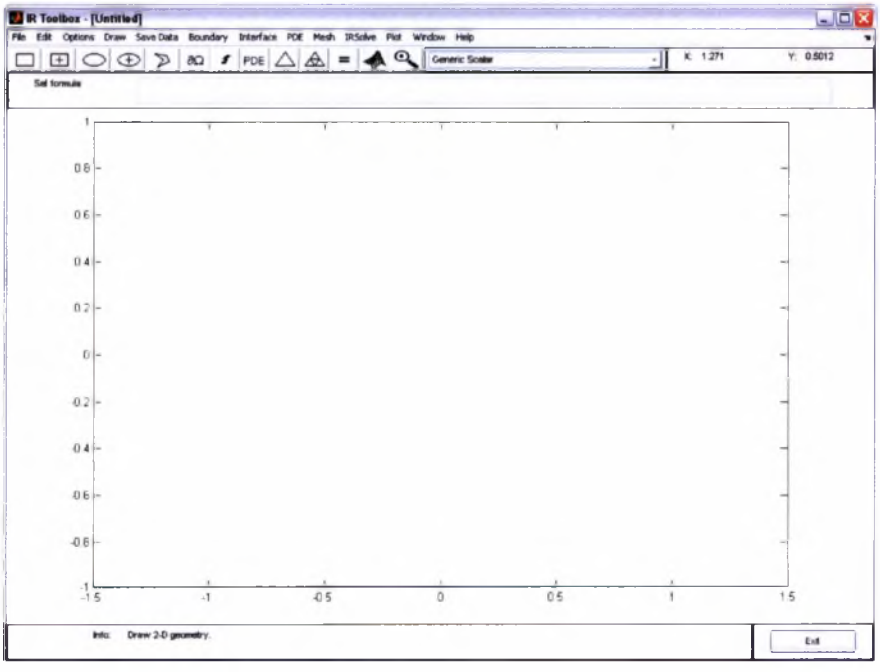
Στο σύστημα προσομοίωσης SciAgent έχουμε τρεις τύπους πρακτόρων: πράκτορας PDECoordinator, πράκτορες PDESolvers και πράκτορες PDEMediators. Ο πράκτορας PDECoordinator ρυθμίζει όλη την εφαρμογή, ο πράκτορας PDEMediator επικοινωνεί με τους πράκτορες που επιλύουν τα προβλήματα σε δύο γειτονικά χωρία και ο πράκτορας PDESolver επιλύει το πρόβλημα.

Εκτός από το λογισμικό που χρειάζεται για την υλοποίηση των παραπάνω πρακτόρων είναι απαραίτητη και η υποστήριξη παρεμβολής (καθώς τα πλέγματα στα γειτονικά χωρία μπορεί να μην συμπίπτουν), μια διαδικασία υπολογισμού των αρχικών τιμών (εξαρτώνται από τις συνθήκες Neumann και Dirichlet), καθορισμός «καλών» τιμών για τις μεθόδους interface relaxation και κατάλληλα κριτήρια για την πραγματοποίηση των επαναλήψεων.

Κεφάλαιο 5: Υλοποίηση εργαλείου IRtool

Το *pdetool*, που είναι υλοποιημένο στο Matlab, παρέχει ένα ευέλικτο περιβάλλον για την μελέτη και την επίλυση των Μερικών Διαφορικών Εξισώσεων. Παρ' όλα αυτά, το *pdetool* δεν μεταχειρίζεται καθόλου τις διεπαφές μεταξύ των υποχωρίων. Αυτό ήταν που μας οδήγησε στην υλοποίηση του *IR Toolbox*. Το Interface Relaxation Toolbox παρέχει ένα γραφικό περιβάλλον για την μελέτη και την επίλυση των μερικών διαφορικών εξισώσεων λαμβάνοντας υπόψη την ίδια στιγμή την μεθοδολογία του interface relaxation. Επομένως, το *IRtool* έχει πρόσθετες ιδιότητες και δυνατότητες. Η βασική ιδέα είναι ότι με το συγκεκριμένο εργαλείο μεταχειριζόμαστε τα τμήματα των διεπαφών και λύνουμε το πρόβλημα χρησιμοποιώντας τις παραμέτρους, όπως είναι η μέθοδος interface relaxation, οι αρχικές τιμές στις διεπαφές και τις συνθήκες σύγκλισης (convergence, tolerance).

Το γραφικό περιβάλλον, που υλοποιήσαμε, διαθέτει ένα μενού για να χειριζόμαστε τα μαθηματικά μοντέλα. Παρακάτω θα παρουσιάσουμε τα περιεχόμενα του μενού και τα παράθυρα διαλόγου ακολουθούμενα από ένα παράδειγμα. Για να ξεκινήσουμε το γραφικό περιβάλλον απλά στην γραμμή εντολών του Matlab την εντολή *IRtool*. Το παράθυρο που εμφανίζεται είναι σαν την εικόνα που εμφανίζεται παρακάτω.

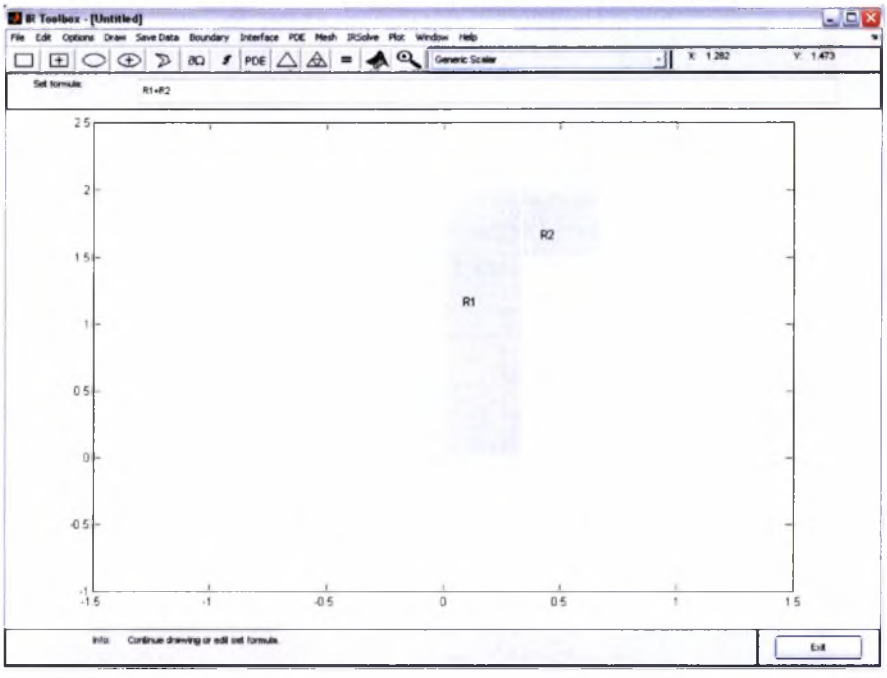


Εικόνα 3. Αρχικό παράθυρο του *IRtool*.

Draw mode

Στο Draw mode έχουμε την δυνατότητα να σχεδιάσουμε το χωρίο του προβλήματος, επιλέγοντας από κάποια αντικείμενα (κύκλος, τετράγωνο, έλλειψη, πολύγωνο). Το μόνο που θα πρέπει να προσέξουμε για να εφαρμόσουμε την μέθοδο του interface relaxation είναι να μην επικαλύπτονται τα χωρία. Κάθε χωρίο που δημιουργούμε έχει ένα ξεχωριστό όνομα και τελικά το χωρίο μας αποτελείται από την ένωση των διαφορετικών υποχωρίων.

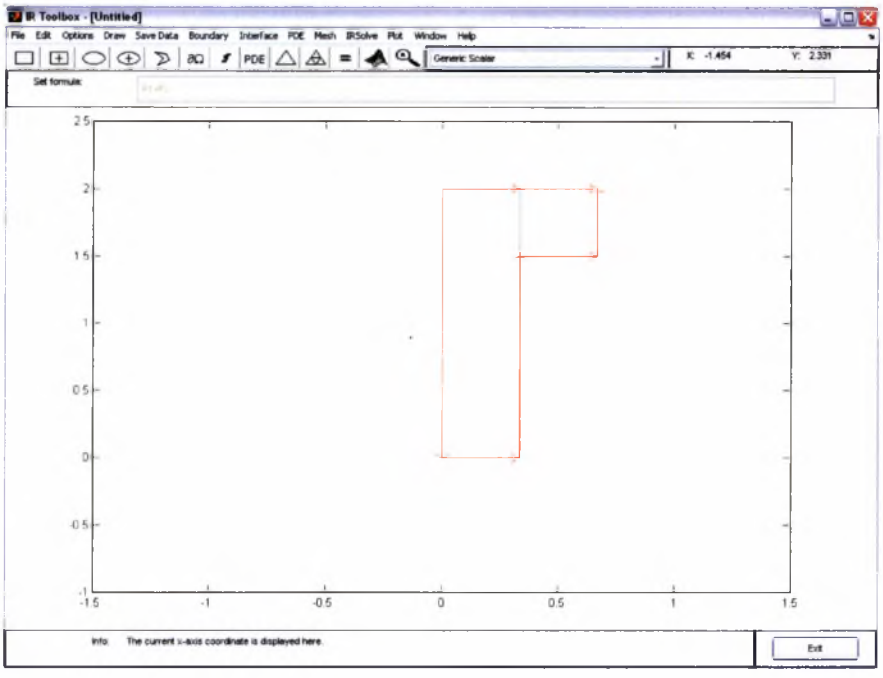
Με αυτό τον τρόπο δημιουργούμε δύο παραλληλόγραμμα για το παράδειγμα μας και το παράθυρο που έχουμε είναι το παρακάτω και το χωρίο που έχουμε είναι το R1+R2.



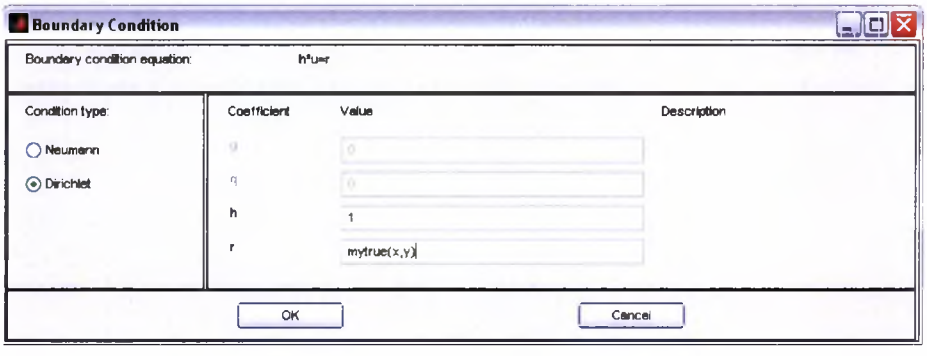
Εικόνα 4. Draw mode.

Boundary mode

Σε αυτό το σημείο μπορούμε να καθορίσουμε τις συνθήκες στα όρια του χωρίου. Τα όρια δηλώνονται με τα χρωματιστά βέλη. Την συνθήκη μπορούμε να την καθορίσουμε από το παράθυρο διαλόγου και μπορεί να είναι ένας απλός αριθμός ή και συνάρτηση. Υπάρχουν δύο τύποι συνθηκών: 1) συνθήκες Dirichlet με u = 0 στο όριο. Οι συνθήκες Dirichlet εμφανίζονται με κόκκινο χρώμα, 2) συνθήκες Neumann (μπλε χρώμα) και 3) μεικτές συνθήκες (πράσινο χρώμα). Ο καθορισμός συνθηκών μπορεί να γίνει μέσα από το παράθυρο διαλόγου.



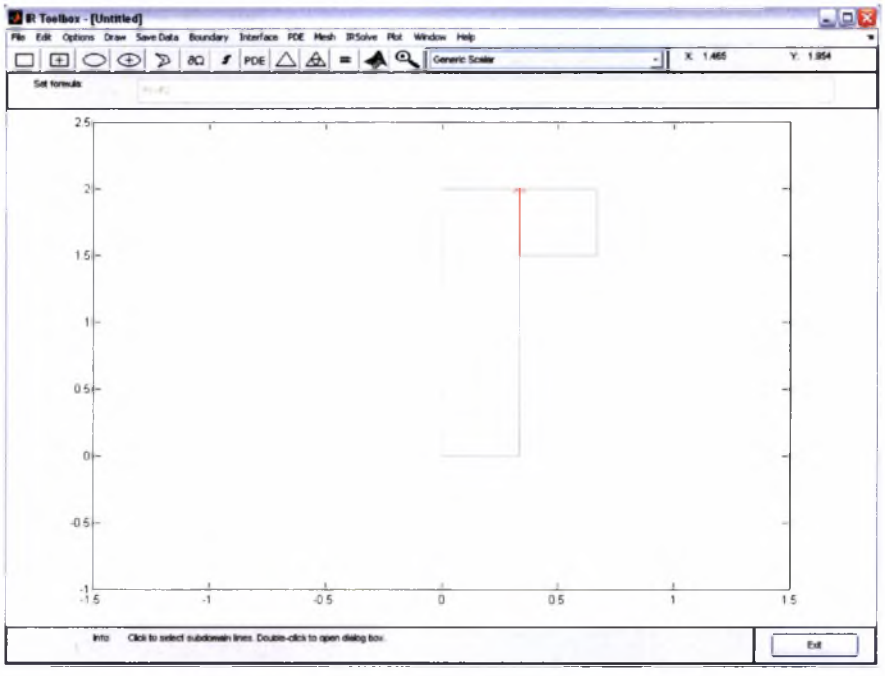
Εικόνα 5. Boundary mode.



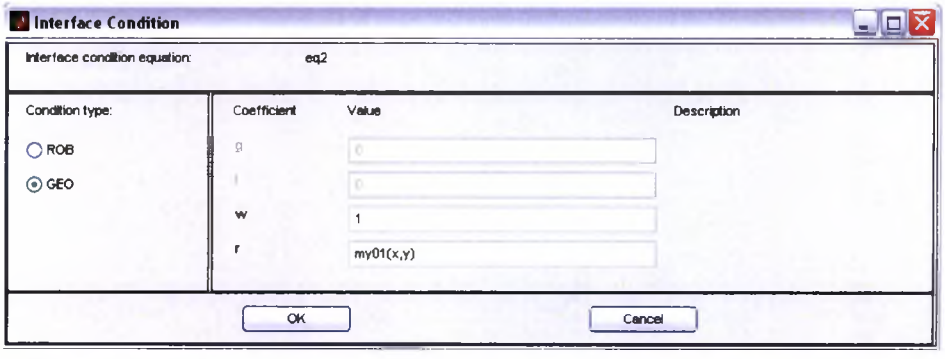
Εικόνα 6. Boundary Condition dialog box.

Interface mode

Ένα αρχικό κομμάτι του εργαλείου *pdetool* που τροποποιήσαμε είναι η δημιουργία του **Interface mode**. Σε αυτό το σημείο εμφανίζονται οι διεπαφές του χωρίου για το πρόβλημα που ορίσαμε. Μέσω του συγκεκριμένου μενού μπορούμε αν θέσουμε τις αρχικές τιμές για την διεπαφή που έχουμε επιλέξει. Στο παράθυρο διαλόγου που εμφανίζεται μπορούμε να επιλέξουμε και την μέθοδο interface relaxation που θα χρησιμοποιήσουμε. Η μέθοδος που έχουμε υλοποιήσει είναι η GEO.



Εικόνα 7. Interface mode.

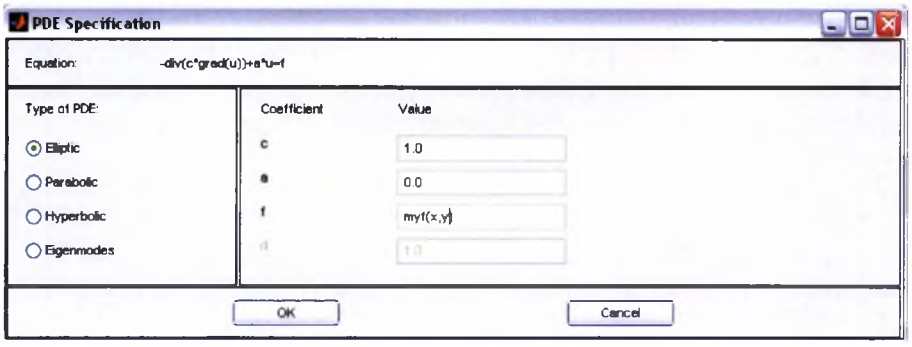


Εικόνα 8. Interface mode dialog box.

Επίσης μπορούμε να καθορίσουμε τις παραμέτρους της μεθόδου από το παράθυρο που ανοίγει αν επιλέξουμε από το μενού Interface Parameters…

PDE mode

Στη συνέχεια, μέσω του PDE mode, μπορούμε να ορίσουμε τους συντελεστές της μερικής διαφορικής εξίσωσης που επιθυμούμε να επιλύσουμε. Οι συντελεστές μπορεί να είναι είτε ένας απλός αριθμός είτε μία συνάρτηση. Επίσης κάθε υποχωρίο μπορεί να έχει διαφορετικές παραμέτρους. Σε αυτό το σημείο μπορούμε ακόμη να επιλέξουμε τον τύπο της διαφορικής εξίσωσης (ελλειπτική, παραβολική, υπερβολή, εξίσωση χαρακτηριστικής ρίζας) Οι τιμές αυτές θέτονται στο παρακάτω παράθυρο διαλόγου.



Εικόνα 9. PDE Specification dialog box

Save Data menu

Το μενού Save Data είναι υλοποιημένο στο *IRtool* στην προσπάθειά μας να αποθηκεύουμε πολλά σημαντικά δεδομένα σε αρχεία στην περίπτωση που τα χρειαζόμαστε. Η βασική ιδέα είναι να διαχωρίσουμε τις πληροφορίες για κάθε χωρίο και να τις αποθηκεύουμε ξεχωριστά.

Πρώτα απ' όλα, διαχωρίζουμε τα δεδομένα της γεωμετρίας σε μικρότερα υποτμήματα *dl1* χρησιμοποιώντας την συνάρτηση *decsg*. Με αυτό τον τρόπο έχουμε διαφορετικούς πίνακες που αντιστοιχούν στα διαφορετικά υποχωρία. Αυτό είναι πολύ πρωταρχικό σημείο καθώς στο *IRtool* θέλουμε να χειριζόμαστε το ολικό πρόβλημα διαφορικών εξισώσεων ως μικρότερα τοπικά προβλήματα και μετά να υπολογίζουμε και να συνθέτουμε τις επιμέρους λύσεις για να καταλήξουμε στην συνολική λύση. Στην συνέχεια, χωρίζουμε τα δεδομένα των συνοριακών συνθηκών, των διεπαφών, των συνθηκών πλέγματος και των συντελεστών των διαφορικών συνθηκών. Τις παραπάνω πληροφορίες τις αποθηκεύουμε σε αρχεία με το όνομα *dom*.m*, όπου m είναι ο αριθμός του υποχωρίου.

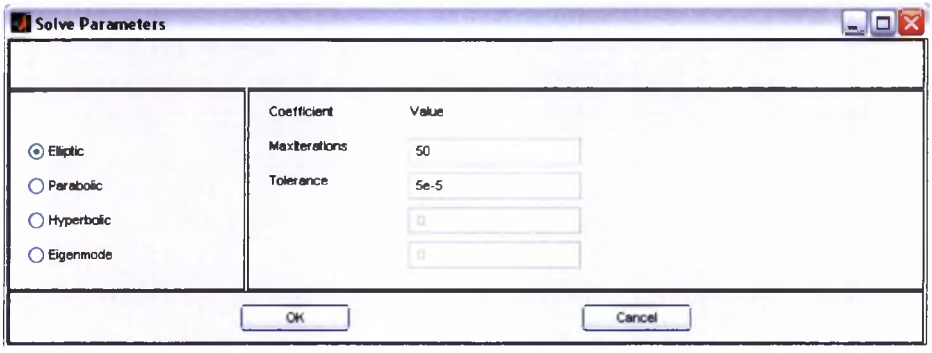
IRSolve menu

Είμαστε πλέον έτοιμοι να λύσουμε το πρόβλημα. Το κύριο τμήμα του *IRSolve* το οποίο υλοποιήσαμε είναι ο παρακάτω αλγόριθμος:

```
for i = 1, 2, . . . Maximum Iterations,
    for j = 1, 2, . . . #domains,
        u, ux, uy = solvedomain j
    end
    for l = 1, 2, . . .  #interfaces,
        IR method (IRflag, l)
    end
end
```

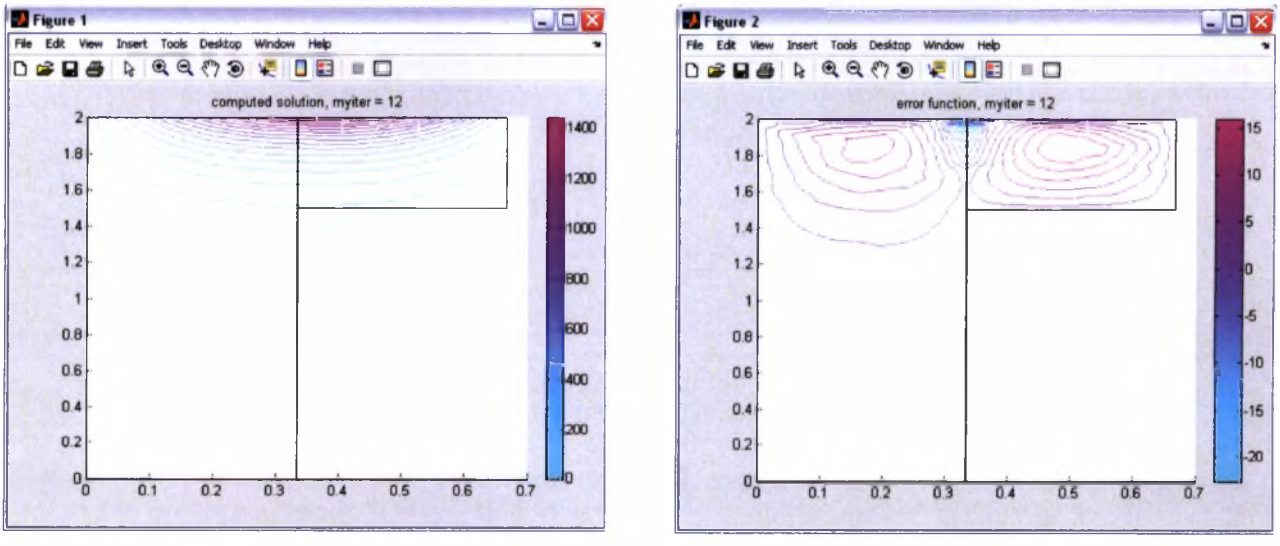
Με την συνάρτηση *solvedomain* εννοούμε πως καλούμε την συνάρτηση *assempde* για να πάρουμε την τιμή *u*, ενώ με τις συναρτήσεις *pdegrad* και *pdeprtni* παίρνουμε τις τιμές των παραγώγων *ux* και *uy*. Η διαδικασία αυτή γίνεται ξεχωριστά και επαναληπτικά για κάθε χωρίο. Για να πάρουμε τις παραπάνω τιμές και τελικά για να πάρουμε την τελική λύση, πρέπει να εφαρμόσουμε την *IRmethod(IRflag, l)* για κάθε διεπαφή. Μέσω αυτής της συνάρτησης καλούμε με βάση το *IRflag* είτε την συνάρτηση ROB, είτε την συνάρτηση GEO. Οι συναρτήσεις αυτές είναι υλοποιήσεις

των μεθόδων interface relaxation. Με αυτό τον τρόπο παίρνουμε τιμές της λύσεις και των παραγώγων για την κάθε διεπαφή. Η διαδικασία αυτή γίνεται επαναληπτικά ανάλογα με τις παραμέτρους (μέγιστος αριθμός επαναλήψεων και ανοχή λάθους) που θέτουμε στο παράθυρο διαλόγου που εμφανίζεται μέσω του μενού του *IRsolve*.



Εικόνα 10. Παράθυρο διαλόγου του IRsolve mode.

Η λύση τελικά του προβλήματος της διαφορικής εξίσωσης τελικά εμφανίζεται σε ένα ξεχωριστό παράθυρο, ενώ παράλληλα εμφανίζεται και ένα παράθυρο με το σφάλμα.



Εικόνα 11. Γραφική παράσταση της λύσης και του λάθους.

A Distributed System for the Simulation of Multi-Domain/Multi-Physics Problems

Thesis Author
Alexandra Bousia

Supervisors
Panagiota Tsompanopoulou
Manolis Vavalis

Department of Computer & Communication Engineering
University of Thessaly, Greece

October 2008

To everyone that stands by me.

Acknowledgments

First, I would like to thank my supervisor, Panagiota Tsompanopoulou, for all the support and the endorsement that she offered me during the past five years of my studies and more specifically during the elaboration of my thesis. Her ideas, her comments and her proposals, and as well as our continuous and endless meetings helped me, motivated me and encouraged me in order to complete my thesis. Without her guidance, this would have been impossible...

Also, I would like to thank my co-supervisor, Manolis Vavalis, who was always next to me, and he was always willing to answer my questions and he was voluntary to advise me with all my wonders. His encouragement helped me during the days that I was disappointed...

Furthermore, I would like to thank all the professors of the Department of Computer & Communication Engineering who helped me during my studies. They all were in their offices willing to offer me information and encouraged me in order to cherish my future profession. Their enlightening collaboration and assistance was critical for my studies. It was a great honor to have met them...

At this point, I would also like to thank my parents and my brother. Especially my father, Nikos, who always was and is here with me and encouraged me to select the Department of Computer & Communication Engineering, the university from which I am about to graduate. My mother, Mary, is always the best advisor, who tells me not to be stressed and that everything is going to work out. A great thank to my brother, Vasilis, for his jokes, his humor and his love and affection...

My studies were balanced with gatherings, walks and excursions with my friends. Their friendship helped me and offered me lasting memories, that I will keep for the rest of my life.

Last but not least, I would like to express my great and tender thanks to Kostas. He always stands by me in every agony and every difficulty that I have to deal with. He is always calm and offers me the balance that is very important and necessary in my life. He is someone who supports me in a very special and unique way...

Table of Contents

Chapter 1: Introduction

The multi-domain/multi-physics problems consist of many different components that have distinct natures, shapes and capabilities. In order to model these systems, we must use parallel computing strategies that treat the systems' components as independent components, because the existing simulation software is only used for simple geometrical shapes. The computation of these systems is achieved by using several methods that have been proposed. The first and the most common computational approach is *domain decomposition* [1]. Domain decomposition refers to geometry discretization by using grids and meshes. The decomposition creates small discrete and inter-connected problems. Each discrete domain has its own equation, but the basic idea is that neighboring components communicate and exchange details and useful information. Another method is *Schwarz splitting.* This approach differs from the domain decomposition because it decomposes the geometry into components with small overlap. Each problem is solved independently and then the global solution can then be computed. The overlap between the discrete domains creates complexity and it is very difficult to solve the Partial Differential Problems by using this method. This has led to the use of another method. The newest non-overlapping approach is the *interface relaxation method.* The domain is decomposed into sub-domains. The sub-domains have different mathematical models and as for the interface between two neighboring components we can use interface conditions that are derived from the physical phenomena (e.g., continuity of mass temperature, conservation of momentum). The models on each sub-domain are solved in the loop of the interface relaxation iteration method to compute the global solution. These methods use one of a variety of "smoothing" formulas to reduce the error in satisfying the interface conditions. These three methods, the domain decomposition, the Schwarz splitting and the interface relaxation method, were proposed in order to handle with different physical models by using parallel computers, but they differ in generality and flexibility. Domain decomposition requires that neighboring components have a lot of shared information about their discretizations. Overlapping Schwarz methods are similarly constrained to a single physical model and also create a tight coupling between neighboring sub-domains. The non-overlapping

Schwarz methods are restricted to a single mathematical model for neighboring sub-domains. The interface relaxation approach imposes no coupling conditions, except those inherent in the mathematical models and it provides maximum generality and flexibility.

The interface relaxation method is the method that we are going to use in order to solve complex PDE problems. The IR methodology is an iterative procedure. First the whole problem is decomposed into smaller and simpler PDE subproblems, where there is no overlap between the neighboring sub-domains. On all sub-domain interface we use initial values, which we estimate. The next step is to solve each single PDE subproblem independently using the estimated values on the interfaces. If the solution that we compute is not the same as the real solution we improve the values on the interfaces using a relaxer. A relaxer is an interface relaxation method, such as GEO, ROB, AVE [2]. This procedure is progressed iteratively until satisfactory accuracy and convergence are achieved.

The main purpose of this thesis is to propose a simulation environment for solving multi-domain/multi-physics problems. More specifically we want to solve elliptic PDE problems that are coupled with both cartesian and general decompositions. The Graphical User Interface (GUI) is a simulation framework where we can draw a 2-D complicated domain and define boundary and interface conditions. We can also specify the partial differential equation, create, inspect and refine the mesh and compute the solution for the particular problem. This framework must have several very desirable properties like fast convergence, wide applicability, increased adaptivity, high efficiency, inherent parallelism and software reuse by integrating advances in different scientific areas like mathematical analysis, numerical analysis, approximation, scientific computing, distributed computing and agent computing. At first we examine only the elliptic PDEs but we believe that our work can be easily extended.

In the Second Chapter, we present two of the Interface Relaxation methods, the GEO method and the ROB method. The Third Chapter is about the software that exists in order to solve the PDE problems. The existing software about the interface relaxation method is presented in the Forth Chapter. In Chapters Five and Six, the computing framework, IRToolbox that we designed and we implemented, is presented.

Chapter 2: PDEs-IR Methods

The domain decomposition methods can be classified into two categories: overlapping and no-overlapping, as we mentioned above. There have already been several studies about the comparison of these two classes. All these methods have been used in order to solve complex problems, nevertheless further research is needed.

We are not going to study the overlapping methods, as they received a great deal of attention over the past years. The non-overlapping and more specifically the interface relaxation methods are discussed. The interface relaxation methods are based on non-overlapping domain decomposition. A complicated PDE problem is split into a set of sub-domains and each subproblem is solved independently, by using initial guesses and smoothing operators on the interfaces. The most known interface relaxation methods are listed below [2].

AVE : A simple method of averaging the solution and its normal derivative along the interfaces.
GEO : A method based on a simple geometric contraction.
NEW : A scheme based on Newton's method to "correct" the interface values.
ROB : An algorithm that uses Robin interface conditions for smoothing.
SCO : A scheme that is based (but not formulated) on a Schur complement approach.
SHO : A method based on the concept of the shooting method for solving Ordinary Differential Equations (ODEs).
SPO : A method originated from the use of Steklov-Poincare operator which involves alternating boundary condition types.

In my thesis one method is selected. This method is GEO Interface Relaxation method [2] and is presented below.

The GEO method.

GEO estimates the new solution for each sub-domain by solving a Dirichlet problem and is classified as a one-step method. The values of the interfaces are obtained by adding to the old ones, a geometrically weighted combination of the normal boundary derivatives of the adjacent sub-domains. In the figure 1 below, we assume that u_L and u_R are the solutions of the PDE problems associated with the left and the right sub-domains, respectively, of the interface point I. We denote by S_L and S_R the right and left slopes at I. As it can be easily seen geometrically, m is the correction that is needed to be added to u_L and u_R so as to match the normal derivatives at I.

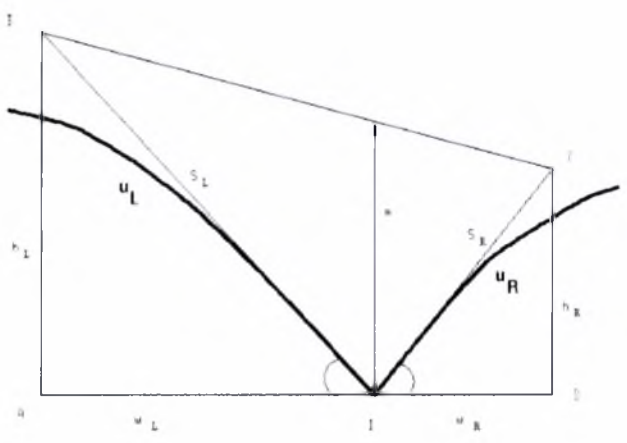


Figure 1. Cross-section perpendicular to the interface where u_L and u_R have slopes S_L and S_R at the interface point I. Changing the values of u_L and u_R by m makes these slopes equal.

To calculate m we consider the two right triangles IAB and CDI (Figure 1) whose heights are given by multiplying the corresponding tangent with the base of the triangle, or by multiplying the normal derivative with the base. The bases w_L and w_R are the widths and can be arbitrarily selected and play the role of the relaxation parameters. The new interface values are now given by adding the weighted average of the heights to the old interface values u_L and u_R. One can intuitively view this as grabbing the function u and I and stretching it up by m until its derivative becomes continuous. Numerical experiments show that the convergence rate does not seem to depend much on the widths. In case that $u_R \neq u_L$ on I we simply use their average.

GEO is given algorithmically by:

for *k=0,1,2,...*

$$ui^{(k+1)} = ui^{(k)} - \frac{w_L w_R}{w_{L+w_R}}\left(\frac{\partial u_L^{(k)}}{\partial x} - \frac{\partial u_R^{(k)}}{\partial x}\right) \textit{ on each interface,}$$
$$u^{(k+1)} = solvepde(ui^{(k+1)}) \textit{ in each sub-domain.}$$

There is a wide class of interface relaxation methods for elliptic differential equations. The methods have many differences. More specifically, the GEO method converge rather rapidly achieving numerical convergence in a small number of iterations, regardless its formulation (multiple steps) and motivation and smoothing techniques (Dirichlet-Neumann conditions).

Chapter 3: Existing software for PDEs

The PDE methodology is implemented on Matlab by simply using the instruction *pdetool* in command line. The Partial Differential Equation (PDE) Toolbox [4] provides a powerful and flexible environment for the study and solution of partial differential equations in two space dimensions and time. The equations are discretized by the Finite Element Method (FEM).

The objective of the PDE Toolbox are to provide you with tools that:

1) Define a PDE problem, e.g., define 2-D regions, boundary conditions, and PDE coefficients,

2) Numerically solve the PDE problem, e.g., generate unstructured meshes, discretize the equations, and produce an approximation to the solution,

3) Visualize the results.

The PDE Toolbox is designed for both beginners and advanced users. The minimal requirement is that you can formulate a PDE problem on paper (draw the domain, write the boundary conditions, and the PDE). By using the command pdetool you can solve any differential equation problem. This invokes the graphical user interface (GUI), which is a self-contained graphical environment for PDE solving. For common applications you can use the specific physical terms rather than abstract coefficients. Using pdetool requires no knowledge of the mathematics behind the PDE, the numerical schemes, or MATLAB. Advanced applications are also possible by downloading the domain geometry, boundary conditions, and mesh description to the MATLAB workspace. From the command line (or M-files) you can call functions from the toolbox to do the hard work, e.g., generate meshes, discretize your problem, perform interpolation, plot data on unstructured grids, etc., while you retain full control over the global numerical algorithm.

The basic equation of the PDE Toolbox that can be solved is the PDE:

$$-\bigtriangledown \cdot (c \bigtriangledown u) + au = f$$

which we shall refer to as the elliptic equation, regardless of whether its coefficients and boundary conditions make the PDE problem elliptic in the mathematical

sense. Analogously, we shall use the terms parabolic equation and hyperbolic equation for equations with spatial operators like the one above, and first and second order time derivatives, respectively. Also, all solvers can handle differential systems.

The PDEs implemented in the toolbox are used as a mathematical model for a wide variety of phenomena in all branches of engineering and science. More specifically the elliptic equations are used for modeling steady and unsteady heat transfer in solids, flows in porous media an diffusion problems, potential flow, electrostatics of dielectric and conductive media.

Additionally, the toolbox can be used for educational purposes as a complement to understanding the theory of the FEM.

A differential equation can be solved by using the PDE Toolbox if we follow the next steps:

Step 1: PDE Problem Definition by using PDE Toolbox

The simplest way to define a PDE problem is using the GUI, implemented in pdetool. There are three modes that correspond to different stages of defining a PDE problem:

- In Draw mode, we can create Ω, the geometry, using the constructive solid geometry (CSG) model paradigm. A set of solid objects (rectangle, circle, ellipse, and polygon) are provided. We can combine these objects using set formulas.

- In Boundary mode, we specify the boundary conditions. You can have different types of boundary conditions on different boundary segments, such as Dirichlet, Neumann and mixed boundary conditions. In PDE Toolbox only the external boundaries are considered. The internal segments are ignored and this is the main objective that we want to change, in order to treat these segments as interface boundaries.

- In PDE mode, we can interactively specify the type of PDE and the coefficients c, a, f, and d. We can specify the coefficients for each sub-domain independently. This may ease the specification of, e.g., various material properties in a PDE model.

Step 2: Solving PDE problems by using PDE Toolbox

Most problems can be solved from the GUI. There are two major modes that help us solve a problem:

- In Mesh mode, we generate and plot meshes. We can control the parameters of the automated mesh generator.
- In Solve mode, we can invoke and control the nonlinear and adaptive solvers for elliptic problems. For parabolic and hyperbolic problems, we can specify the initial values, and the times for which the output should be generated. For the eigenvalue solver, we can specify the interval in which to search for eigenvalues.

After solving a problem, we can return to the Mesh mode to further refine our mesh and then solve again. We can also employ the adaptive mesh refiner and solver. This option tries to find a mesh that fits the solution.

For advanced, nonstandard applications we can transfer the description of domains, boundary conditions etc. to MATLAB workspace. From there we use the functions of the PDE Toolbox for managing data on unstructured meshes. We have full access to the mesh generators, FEM discretizations of the PDE and boundary conditions, interpolation functions, etc. We can design our own solvers or use FEM to solve subproblems of more complex algorithms.

Step 3: Visualizing the results

From the graphical user interface we can use Plot mode, where we have a wide range of visualization possibilities. We can visualize both inside the pdetool GUI and in separate figures. We can plot three different solution properties at the same time, using color, height, and vector field plots. Surface, mesh, contour, and arrow (quiver) plots are available. For surface plots, we can choose between interpolated and flat rendering schemes. The mesh may be hidden or exposed in all plot types. For parabolic and hyperbolic equations, we can even produce an animated movie of the solution time dependence. All visualization functions are also accessible from the command line.

Basics of the Finite Element Method

The solutions of simple PDEs on complicated geometries can rarely be expressed in terms of elementary functions. We are confronted with two problems: First we need to describe a complicated geometry and generate a mesh on it. Then we need to discretize your PDE on the mesh and build an equation for the discrete approximation of the solution. The pdetool graphical user interface provides us with easy-to-use graphical tools to describe complicated domains and generate triangular meshes. It also discretizes PDEs, finds discrete solutions and plots results. We can access the mesh structures and the discretization functions directly from the command line (or M-file) and incorporate them into specialized applications.

We start by approximating the computational domain with a union of simple geometric objects, in this case triangles. The triangles form a mesh and each vertex is called a node. We are in the situation of an architect designing a dome. He has to strike a balance between the ideal rounded forms of the original sketch and the limitations of his simple building-blocks, triangles or quadrilaterals. If the result does not look close enough to a perfect dome, the architect can always improve his work using smaller blocks.

The solution should be simple on each triangle. Polynomials are a good choice: they are easy to evaluate and have good approximation properties on small domains. We can ask that the solutions in neighboring triangles connect to each other continuously across the edges and we can still decide how complicated the polynomials can be. Just like an architect, we want them as simple as possible. Constants are the simplest choice but you cannot match values on neighboring triangles. Linear functions come next. This is like using flat tiles to build a waterproof dome, which is perfectly possible.

Chapter 4: Existing software for Interface Relaxation

The Interface Relaxation Methodology is a new area of study and that's why there are only a few implementations that cover this research. In this chapter, we are going to present the implementations that were proposed during the past decades. A first but naive implementation was in 1991 [5]. It was based on TCP/IP routines. A prototype implementation was proposed by Drashansky (Networked Agents for Scientific Computing) and is based on a Multidisciplinary Problem Solving Environment. With this environment, he managed to solve complex problems. Another implementation was recommended by McFaddin (RELAX). This system can support high-level interfaces and an object-oriented framework for sets of problem solving modules. Last but not least, we contemplate the SciAgent implementation. This implementation is based on domain decomposition and the parallel solution of each problem by using the characteristics of the Interface Relaxation methods.

Networked Agents for Scientific Computing.

Drashansky's implementation [6], [7], comprises a simulation environment that solves multi-physics/multi-domain problems. This implementation is based on a Multidisciplinary Problem Solving Environment (MPSE) that is a software kernel which deals with tailored, flexible complex problems. This environment takes advantage of the characteristic of the physical world. This characteristic supposes that a physical complex problem consists of a set of simple, heterogenous, connected problems and domains that have different behaviors and shapes and they interact through geometric and physical interfaces with each other. On the boundaries and the interfaces, several models are used in order to represent the constraint conditions and mathematical formulas, such as partial differential equations are used for the representation of the problems on each domain. As we understand in a MPSE, the complete functional (mathematical) description of

the problem includes: 1) definition of the sub-domains, 2) definition of the models in each sub-domain (these models are defined by the user) and 3) definition of the interfaces between sub-domains and physical conditions along them. Taking into account all the above information, we conclude that the MPSE software framework deals with a wide variety of problems in high quality. The Drashansky implementation is based on the use of two agents [8]: the solver agent and the mediator agent. Each solver agent is used to compute the local solution of a subproblem in a sub-domain of the global problem. From the other hand the mediator agent is responsible for dealing with the interface between two neighboring sub-domains. Sometimes, there may be more than one mediator agents for one interface and each mediator agent operates on a separate part of the whole interface. The mediator agent computes data on the interface by using some specific methods and sends these information to the solvers agents. So, the solvers agents and the mediator agents form a network that solves the global problem. The architecture of this implementation is shown below in Figure 2.

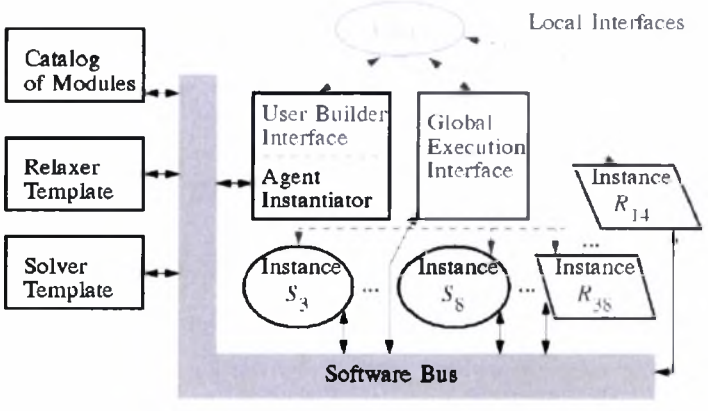


Figure 2. MPSE Architecture.

As we see on the above figure, the mediator and the solver agents communicate through a *software bus* and they have their own local user interface in order to interact with the user. If the solver agent does not have all the data that he needs in order to compute the local solution, he waits the mediator agent to send him the missing but useful data. The mediator agent waits if a solver on either side of the interface is missing. This kind of synchronization is very important [9].

Also, an MPSE for simulating composite PDE models is the SciAgents which is discussed below.

RELAX.

McFaddin proposed RELAX [5], [10], an experimental system for using collaborating PDE solvers. From the computational science point of view RELAX is a prototype of a new problem solving methodology for complex PDE problems. In this implementation, the whole geometry is decomposed into blocks and each block is characterized by a mathematical formula. The solution is computed for every block is displayed directly and passed to the next procedure.

This system can support high-level interfaces and an object-oriented framework for sets of problem solving modules. The problem solving modules are interact only through the RELAX framework. These objects have their own numerical methods, their own editors to interact with users, their own display capabilities, etc. Some objects may, of course, be clones of a single master object. The RELAX system allows many master objects including those created by the user on the spot by combining and/or specializing existing objects. Its capabilities are the following:

- Geometry: One can create collections of building block shapes (sub-domains) to define a complex geometric object (domain). The basic shapes have parameters (e.g., width, rotations) to help shape the composite object.
- PDEs: One can define a different partial differential equation and associated boundary conditions on each sub-domain.
- Solvers: The building blocks also have associated PDE solving methods (in principle, one could have several such methods) whose parameters (e.g., mesh size) can be specified.
- Interfaces: As the sub-domains are assembled, interfaces are created and explicitly identified as objects in the RELAX system.
- Relaxers: Interface relaxation formulas are assigned to each interface. These may be written by the user or selected from a menu of predefined formulas.
- Schedules: An ordering of applying PDE solvers on the sub-domains and the relaxers on the interfaces constitutes a schedule. This ordering may be a simple algorithm (e.g., Round-Robin) selected from a menu or interactively specified step by step.

SciAgent.

SciAgent implementation consists of a whole class of Interface Relaxation methods in an agent based framework that is implemented mainly using C and JAVA. This implementation is used for general two-dimensional decompositions of linear and non-linear elliptic PDE problems. The SciAgents [2] exploit the inherent parallelism in the interface relaxation methods using the Agents computing paradigm over a network of heterogenous workstations. The components of a SciAgent system are shown in Figure 3 below.

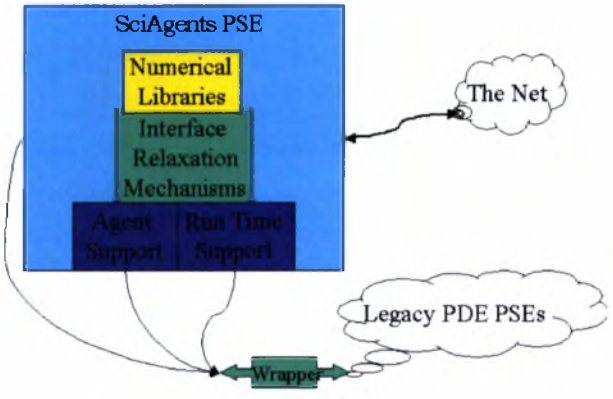


Figure 3. Components of the SciAgent system.

Specifically, the SciAgents transform the physical problem into a network of local PDE solvers and interface relaxers. In the SciAgent prototype there are three types of agents: one PDECoordinator agent, several PDESolver and PDEMediator agents. The PDECoordinator, as it is figured by its name, is responsible for the control of the entire application and coordinate the whole procedure, a PDEMediator arbitrates between the two solvers sharing a boundary between two domains, and a PDESolver is a wrapper for the legacy application and solves the local problem.

When the PDECoordinator agent is started, reads a problem description file and writes the information into its model. The input file contains information about the number of solvers and mediators, the characteristics of the interfaces, the initial guesses on the interfaces, the interface relaxation methods and the

names of the machines that will be used to solve the whole problem. The next step in the procedure is to create and configure the PDESolver and PDEMediator agents. The PDECoordinator uses their addresses to setup the communication between them. Then the coordinator waits for messages from the mediators, regarding the status of the convergence to the solution of the problem, or from the user. The messages from the user are to change the values of specific variables of the input file, such as convergence tolerance or to force the execution to stop.

The PDESolver agent is responsible to solve the problem locally and path the input/ output files etc. In the first state, the PDESolver starts-up the Pelltool which compiles the .e file that describes the local PDE problem, and creates the executable that will be used later on by the ExecuteTool. Both Pelltool and ExecuteTool are parts of PELLPACK system [2]. In the next state, the solver extracts the points on the interfaces from the file that contains the mesh/ grid points, and writes them into a file. Then the solver notifies the mediators that the files are ready. The PDESolver agent remains idle until being notified by the mediator that the list with all the points and their initial guesses are stored in a file at a specific location. Then the solver uses these files to run the ExecuteTool to solve the problem and then, the solver send a message to the mediators that new values are computed, and waits for their response. Depending on the message from the mediators, the solver will solve the problem again, remain idle waiting for the other solvers, or plot the local solution. The PDECoordinator is able to terminate the PDESolver by sending an appropriate message.

The mediator agent, PDEMediator, is created and configured by the PDECoordinator agent. The mediator agent has a complete description of the interface, the relaxation method used, the solvers to collaborate with, the location of the input/ output files, the location of the legacy programs, the tolerance used to decide convergence, and the initial guess function. This information is provided by the coordinator agent. After being started, the mediator waits for the boundary points from the neighboring solvers. In the next stage, the mediator combines the two point lists and then uses the initial guess to compute values at these points. Afterwards, the mediator sends a message to the two solvers that the files with the points and their values are ready. The mediator remains idle, waiting for new values from the two solvers. When it receives new values it moves to the next stage, reads the new data and compares them with current data. Then the mediator agent uses the relaxation method to calculate the new values for the

boundary conditions. If convergence is reached on this interface then the mediator sends messages to the solvers and informs the PDECoordinator about the local convergence so it will be able to decide on global convergence. A message from the coordinator is sent to the mediator and the PDEMediator will finish, in case of global convergence or wait for new data from the two solvers.

SciAgents require strong interpolation support, procedure for estimating initial guesses, mechanisms for determining "good" values for relaxation parameters and criteria to control the iterative procedure. Interpolation is needed because grids/ meshes do not necessarily match on the interface segment. Also the estimation of initial values on the interface is a very complex procedure, because the Neumman and the mixed boundary conditions are sensitive to their initial guesses and as the PDE problems get more complicated better initial guesses will be needed.

Chapter 5: IRToolbox Implementation

The IRToolbox is implemented in Matlab and we used the files of *Pdetool* in order to add some extra capabilities. In order to achieve that we modified some of the existing files, we created new ones and we left other unchanged.

We begin with the file *IRtool.m* which opens the toolbox GUI, when we call *IRtool* with no arguments. We will explain every change that we've made in this file and we will present each new application that we casted. Now, we are going to indicate the basic changes. The Interface menu and the IRsolve menu are the menus that we have change in order to add new application and capabilities in the *pdetool*.

Interface mode

In order to enter the interface mode we added a pull-down menu and a button icon. For the pull-down menu we created an inter_menu. In this menu, we provide the sub-menus: **Interface Mode**, **Specify Interface Conditions** and **Interface Parameters...** by using the functions *menulabel* and *uimenu*. Also, we used the function *pdeicon* in order to load the button icons that we need, in our case we created the icon for interface mode. The *pdeicon* function provides the icon library for the PDEToolbox and the IRToolbox. When the user selects **Interface mode** from the menu or pushes the icon ***I***, automatically a callback function is activated. This function is *IRtool('IRmode')*. In this Callback function, we first get the *User Data* that appear in the *pde_fig* which is the main figure of *IRtool*. If there is no geometry in the figure, *IRtool('2domain')* function is called. This secondary function *IRtool('IRmode')* creates a geometry by using the functions *pderect*, *pdepoly*, *pdeellip* and *pdecirc*. This geometry is an example geometry. If we open the *IRtool* and the main window is empty, we can use the example geometry of the *IRtool* by simply choosing **Interface mode** for the **Interface** menu. The example geometry that we use is the geometry that is presented below (Uniform Case). It consists of three rectangles. As we continue in the function *IRtool('IRmode')* the basic idea is to deactivate everything else and activate only interface lines. This is enabled by the use of the callback function *IRtool('drawIR')*. This function extracts the User Data about the geometry

with the command *dl1 = getappdata(pde_fig, 'dl1')*. Then, we find the interface segments with the line *intbounds=find([dl1(6,:) ~=0 dl1(7,:) ~=0])*, because we know that a segment is an interface line when it adjoins with two domains from both sides. The left and the right domain of an interface are shown in the sixth and the seventh line of the matrix *dl1* and are represented by the number of each domain. Therefore, when none of the items *dl1(6,i)* and *dl1(7,i)* is 0, this means that i is an interface segment. At this time, we also make the interface lines into arrows in order to indicate direction. after this procedure, we have the geometry designed and the interface lines indicated. The default color is red which symbolizes the GEO method that is chosen.

So, in interface mode, we are ready to select an interface line and modify its parameters. This is easy if we double-click on the interface a callback function named *IRtool('boundclkIR',1)* is activated. In addition, this function calls the *IRtool('set_IR')* function. Now we can set the interface parameter values, because when we double-click on an interface and because of the *IRtool('set_IR')* an external function is called. This function is *IRpdebddlg*, which manages the interface condition dialog box for the *IRToolbox*. First of all, it brings up and initializes the dialog box. In the dialog box, we can choose between *ROB* and *GEO*. When we choose *GEO* the value 1 is set in the variable *IRflag*, which is a very important variable for the rest of the program, because it distinguishes the methods. If we choose *ROB*, *IRflag* is set to 2. Also in the dialog box, we can type the initial guess for the interface that we have selected. This can be done by simply typing the preferable function in the parameter w. The procedure is terminated if we press the OK button.

The information about the interfaces is saved in a file called *pdebound*. This file can be used to specify the boundary and interface conditions of a PDE problem. *[q, g, h, r] = pdebound(p, e, u, time)* produces values of the boundary conditions. The matrices p and e are mesh data. From the matrix e we only need a subset of the edges in the mesh. The input arguments u and *time* are used for the nonlinear solver and time stepping algorithms respectively. The matrix u is the solution that we get from *assempde*. q and g must contain the value of the matrices q and g on the mid point of each boundary or interface. Thus we have *SIZE(q) = [N^2 NE]*, where N is the dimension of the system, and NE the number of edges in e, and *SIZE(g)=[N NE]*. For the Dirichlet or GEO case, the corresponding values must be zeros. h and r must contain the values of the matrices h and r at the

first point on each edge followed by the value at the second point on each edge. Thus we have *SIZE(h)=[N^2*NE]*, where N is the dimension of the system, and NE the number of edges in e, and *SIZE(R)=[N 2*NE]*. When M<N, h and r must be padded with N-M rows of zeros. The *pdebound* file is frequently used in our program.

Back in *IRtool* file, from the **Interface** menu, we can select the sub-menu **Interface Parameters...**, which activates the callback function *IRtool('IR_param')*. This function and the function *pdedlgIRparam*, which is defined in a separate file, are responsible for the dialog box that opens. In this dialog box, we can define the parameter *omega* for the *GEO* method and *lamda* for the *ROB* method. These parameters are saved in handles because we need these parameters to be visible from all files. The handles data can be extracted by using the functions *getappdata* and *setappdata*. The function *value = getappdata(h,name)* gets the value of the application-defined data with the name specified by *name*, in the object with the handle *h*. If the application-defined data does not exist, MATLAB returns an empty matrix in value. The function *setappdata(h,name,value)* sets application-defined data for the object with handle *h*. The application-defined data, which is created if it does not already exist, is assigned a *name* and a *value*. Value can be any type of data. *pdedlgIRparam* manages the interface parameters specification dialog box. These are the changes that we have made in the **Interface** menu.

IRsolve mode

The other basic part of *pdetool* that we have modified is the menu **Solve**. If we press **Parameters** from the **IRSolve** menu, the callback function IRtool*('IRSolve_param')* and the function *pdedlgIR* are activated. This file brings up the IRSolve specification dialog box. In this dialog box, we can modify the parameters for the solution of PDE. These parameters are maximum number of iterations and tolerance. We have saved these values in handles, in order to make it easy to access them. We again use the unction *getappdata* and *setappdata* in order to access the data that we have saved in handles.

Back to the file *IRtool*, we can choose **IRSolve PDE** from the menu **IRSolve** or press the "=" button, in order to solve the PDE problem that we have specified. This is the basic part of our problem, as after having determined the interface conditions, the interface parameters and the IRSolve parameters, we are now ready to explain how we managed to solve the PDE problem. The

callback function that computes the solution is *IRtool('IRsolve')*. At this point, we can decompose the information about the sub-domain, as we want to solve each domain alone and independently and then compose the distinct solutions in order to get the whole solution of the geometry. First of all, we must extract the decomposed list, that represents the geometry data. This can easily be done by the use of the following commands:

gd = get(findobj(get(pde_fig,'Children'), 'flat', 'Tag', 'PDEMeshMenu'), 'UserData');
and [dl1_all, bt1, pdedl, bt, msb_all] = decsg(gd);.

The function *decsg* decomposes the solid geometry into minimal regions. *[DL, BT, DL1, BT1, MSB1]=DECSG(GD)* decomposes the solid objects *GD* into the minimal regions *DL*. The solid objects are represented by the Geometry Description matrix, and the minimal regions are represented by the Decomposed Geometry matrix. *DECSG* returns all the minimal regions. It also returns a Boolean table *B*T that relates the original solid objects to the minimal regions. A column in *BT* corresponds to the column with the same index in *GD*. A row in *BT* corresponds to the column with the same index in *DL*. A second set of minimal regions *DL1* with a corresponding Boolean table *BT1* is the set of minimal regions that have a connected outer boundary. These minimal regions can be plotted by using MATLAB patch objects. The calling sequences additionally returns a sequence *MSB1* of drawing commands for each second minimal region. At this point, it is very important to give the description of the geometry matrix. The Geometry Description matrix *GD* describes the solid model that we draw in the *IRTOOL* GUI. Each column in the Geometry Description matrix corresponds to an object in the solid geometry model. Four types of solid objects are supported. The object type is specified in row one:

1. For the circle solid, row one contains 1, the second and third rows contain the x- and y-coordinates of the center respectively. Row four contains the radius of the circle.

2. For a polygon solid, row one contains 2, and the second row contains the number, N, of line segments in the boundary. The following N rows contain the x-coordinates of the starting points of the edges, and the following N rows contain the y-coordinates of the starting points of the edges.

3. For a rectangle solid, row one contains 3. The format is otherwise identical to the polygon format.

4. For an ellipse solid, row one contains 4, the second and third row contain the center x- and y-coordinates respectively. Row four and five contain the major and minor axes of the ellipse. The rotational angle of the ellipse is stored in row six.

The Decomposed Geometry matrix *DL* contains a representation of the minimal regions that have been constructed by the *DECSG* algorithm. Each edge segment of the minimal regions correspond to a column in *DL*. In each such column rows two and three contain the starting and ending x-coordinate, and rows four and five the corresponding y-coordinate. Rows six and seven contain left and right minimal region numbers with respect to the direction induced by the start and end points (counter clockwise direction on circle and ellipse segments). There are three types of possible edge segments in a minimal regions:

1. For circle edge segments row one is 1. Rows eight and nine contain the coordinates of the center of the circle. Row 10 contains the radius.

2. For line edge segments row one is 2.

3. For ellipse edge segments row one is 4. Rows eight and nine contain the coordinates of the center of the ellipse. Rows 10 and 11 contain the major and minor axes of the ellipse respectively. The rotational angle of the ellipse is stored in row 12.

So, when we have the decomposed geometry data, that is named *dl1_all*. Next we have to get this matrix for each of the sub-domains. We achieved that by using the command *[dl1,bt1,pdedl,bt,msb]=decsg(gd(:,j));* , where j is the identification number of the corresponding domain. Afterwards, by using the matrix *msb* that is also produced from *decsg* we create another matrix also called *msb*. This matrix contains the indexes of *dl1_all* that correspond to the boundaries of each domain. By this way, we can extract the boundary data (matrix *bl*) for each distinct domain. With the following commands, we manage to get all the information we need for both the boundaries and the interfaces.

bl_all=get(findobj(get(menuhndl,'Children'),'flat', 'Tag','PDEBoundMode'), 'UserData');

bl_all_inter=get(findobj(get(menuhndl,'Children'),'flat', 'Tag','PDEInterMode'), 'UserData');

bl1 = bl1_all(:,msb(:,1));

The value of the interface in the matrix *bl1* is determined from the Interface Condition dialog box as the function *my01(x,y)* for the first interface, *my02(x,y)*

for the second and e.t.c. In our code, we change this information into *ir01(x,y)* and *ir02(x,y)* for the first interface in the left and in the right domain respectively and into *ir03(x,y)* and *ir04(x,y)* for the second interface and e.t.c.

Furthermore, it is very important to extract the mesh conditions(p, e, t) for each domain. This is possible by using the line *[p, e, t] = initmesh(dl1,'hmax', trisize);*, where *trisize* is a parameter that specifies the maximum edge size. *INITMESH* builds an initial PDE triangular mesh. *[P,E,T]=INITMESH(G)* returns a triangular mesh using the geometry specification function G. It uses a Delaunay triangulation algorithm. The mesh size is determined from the shape of the geometry. G describes the geometry of the PDE problem, as we mentioned before. For more details we can check the function *PDEGEOM*. The outputs P, E, and T are the mesh data. In the point matrix P, the first and second rows contain the x- and y-coordinates of the points in the mesh. In the edge matrix E, the first and second rows contain the indices of the starting and ending point, the third and fourth rows contain the starting and ending parameter values, the fifth row contains the boundary segment number, and the sixth and seventh rows contain the left- and right-hand side sub-domain numbers. In the triangle matrix T, the first three rows contain indices to the corner points, given in counter clockwise order, and the fourth row contains the sub-domain number. The *hmax* parameter controls the size of the triangles on the mesh. *INITMESH* creates a mesh where no triangle side exceeds *hmax*.

In addition, we have to extract the PDE coefficients c, a, f and d with the following lines:

```
params=get(findobj(get(pde_fig,'Children'),'Tag','PDEMenu'),'UserData');
ns=getappdata(pde_fig,'ncafd');
nc=ns(1); na=ns(2); nf=ns(3); nd=ns(4);
c_all=params(1:nc,:);
a_all=params(nc+1:nc+na,:);
f_all=params(nc+na+1:nc+na+nf,:);
d_all=params(nc+na+nf+1:nc+na+nf+nd,:);
```

and after having extracted the coefficients for all the domains, then we easily extract the coefficients of each individual domain.

We save all the above information in cell arrays. We create a cell array for each domain and each discrete cell array contains the decomposed geometry matrix, the boundary matrix, the mesh conditions and the PDE coefficients of

each domain.

The next step is to create a matrix, named *inter*. The matrix *inter* has rows as many as the interfaces that exist in our geometry. In the first column, there is the identification number of the interface. In columns 2 and 4, we save the numbers of the domain that adjoin with the interface, and in columns 3 and 4 we have the identification numbers of the interface in each domain. We use this matrix in many cases, so we save it into handles, so it is visible from all files.

Now that all the information for each domain is available, we are ready to solve the PDE problem. First of all, we have the main loop that is executed iteratively based on the parameters: maximum number of iterations and tolerance. Then we have a first loop among the domains. This loop computes the solution u and the derivatives ux and uy, by using the functions *assempde*, *pdegrad* and *pdeprtni*. *ASSEMPDE* assembles the stiffness matrix and right hand side of the PDE problem. More specifically, *U=ASSEMPDE(B,P,E,T,C,A,F)* assembles and solves the PDE problem that is defined by the equation *-div(c*grad(u))+a*u=f*, on a mesh described by P, E, and T, with boundary conditions given by the function named B. It eliminates the Dirichlet boundary conditions from the system of linear equations when solving for U. In the first iteration, the boundary conditions that we use, calls the functions *ir*(x,y)*, where * represents the identification number of each file as we discussed above. In this file *ir*(x,y)* as long as the number of iteration is 0, the interface condition has the initial guess that we type in file *mytrue(x,y)* in the interface dialog box. On the other hand, when the number of iterations increases, in the file *ir*(x,y)*, we have the value that is produced from the Interface Relaxation method that we use. Back to *assempde*, for the scalar case the solution u is represented as a column vector of solution values at the corresponding node points from P. For a system of dimension N with NP node points, the first NP values of U describe the first component of u, the following NP values of U describe the second component of u, and so on. Thus, the components of u are placed in the vector U as N blocks of node point values. B describes the boundary conditions of the PDE problem. B can either be a Boundary Condition Matrix or the name of Boundary M-file (For more details, we can see the file pdebound). The coefficients c, a and f of the PDE problem can be given in a wide variety of ways:

- A constant.
- A row vector of representing values at the triangle centers of mass. A

MATLAB text expression for computing coefficient values at the triangle centers of mass. The expression is evaluated in a context where the variables *X, Y, SD, U, UX, UY*, and *T* are row vectors representing values at the triangle centers of mass. (T is a scalar). The row vectors contain x- and y-coordinates, sub-domain label, solution with x and y derivatives and time. *U, UX*, and *UY* can only be used if *U0* have been passed to *assempde*. The same applies to the scalar *T*, which is passed to *assempde* as *TIME*.

- A sequence of MATLAB text expressions separated by exclamation marks "!". The syntax of each of the text expressions must be according to the above item. The number of expressions in the sequence must be equal to the number of sub-domain labels in the triangle list t.

- The name of a user-defined MATLAB function that accepts the arguments *(P,T,U,TIME)*. *P* and *T* are mesh data, *U* is the *U0* input argument and *T* is the *TIME* input argument to *assempde*.

If *C* contains two rows with data according to any of the above items, they are the c(1,1), and c(2,2), elements of a 2-by-2 diagonal matrix. If c contains three rows with data according to any of the above items, they are the c(1,1), c(1,2), and c(2,2) elements of a 2-by-2 symmetric matrix. If C contains four rows with data according to any of the above items, they are the c(1,1), c(2,1), c(1,2), and c(2,2) elements of a 2-by-2 matrix.

The next thing that we do is to compute the values *u* on each interface. This is accomplished by the use of the command *[unew1 unew2 u1is x1is y1is x2is y2is] = IRmethod(IRflag,inter(1,1))*; *IRmethod* calls the function *GEO* or *ROB* based on the value of *IRflag*. The variables *unew1* and *unew2* are global. The function GEO is executed in proportion with the identification number of the interface. According to this number, in *GEO* function we extract the mesh information for the corresponding domains and the values of *u*, *ux* and *uy*, as well. We use interpolation in order to get the values of the derivatives of the two domains on the nodes and finally we compute the value of the variables *unew1* and *unew2*. We also produce figures for the history of interface values and the corresponding error. Back to function IRtool, we present the figures of the whole solution and the error of our PDE problem.

Chapter 6:
IRTool Graphical User Interface

The PDE Toolbox which is implemented in Matlab provides a flexible environment for the study and solution of partial differential equations. Nevertheless, it does not handle with interface boundaries in a proper manner. This is the reason that inducts us to implement the IR Toolbox. The Interface Relaxation Toolbox provides a graphical user environment for the study and solution of partial differential equations as it considers at the same time the interface relaxation methodology. So, the IRToolbox has some additional properties and capabilities. The basic idea was that with the IRToolbox we can deal with the interface segments and solve the global problem by contemplating parameters such as interface relaxation method, initial guesses on the interface segments and tolerance (convergence).

The Graphical User Interface (GUI) has a pull-down menu bar that we can use to control the modeling. It conforms to common pull-down menu standards. Menu items followed by a right arrow lead to a sub-menu. Menu items followed by an ellipsis lead to a dialog box. Stand-alone menu items lead to direct action. Some menu items can be executed by using keyboard accelerators. *IRtool* also contains a toolbar with icon buttons for quick and easy access to some of the most important functions. The following sections describe the contents of *IRtool* menus, the dialog boxes associated with menu items and an illustrated example.

To get started with the graphical environment we simply type in the Matlab prompt the command *IRtool*. The GUI looks like the figure below (Figure 4).

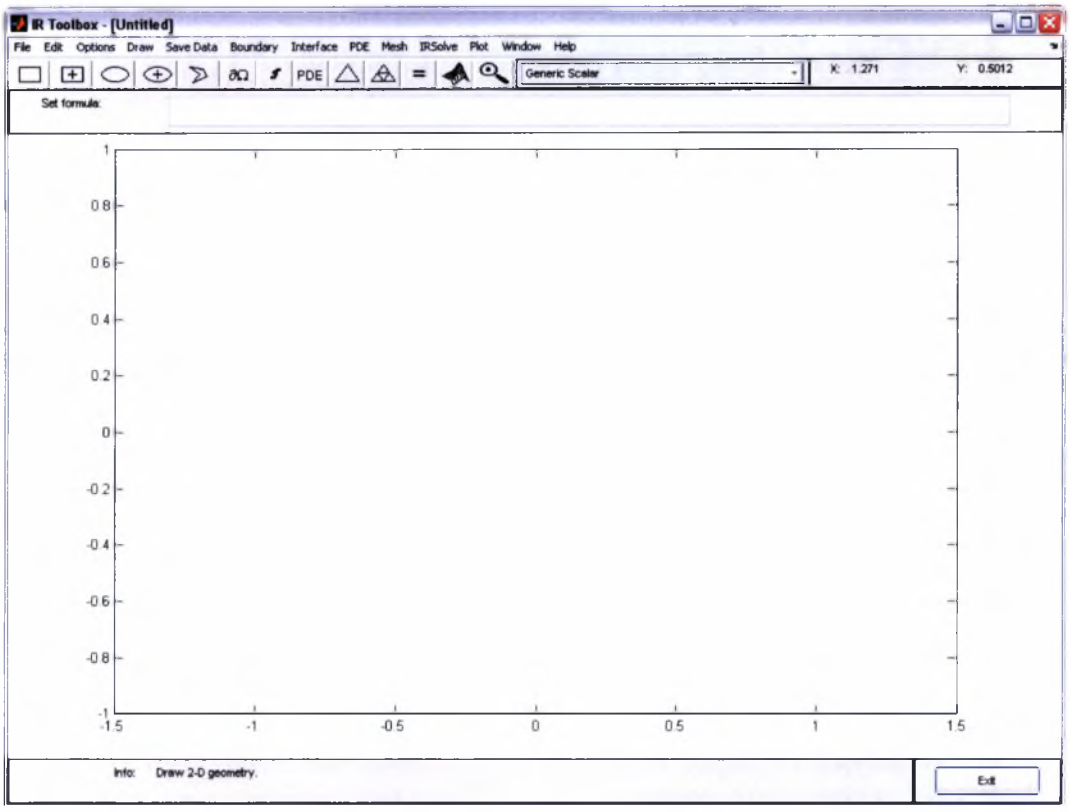


Figure 4. The main window of *IRtool*.

Now, we are going to explain the menus of the IRtool. Some of the menus are the same as in Matlab *pdetool* an d so, we are going to mention them briefly.

File Menu

The pull-down menu has the following applications: *New*, *Open*, *Save*, *Save As*, *Print*, *Exit*. These applications deal with basic procedures that exist in many toolboxes of Matlab.

Edit Menu

The **Edit** menu provides the applications below: *Undo*, *Cut*, *Copy*, *Paste*, *Clear*, *Select All*.

Options Menu

Similarly, the **Options** menu is the same as the pull-down **Options** menu of the *pdetool*. Its applications are: *Grid*, *Grid spacing*, *Snap*, *Axis Limits*, *Axis*

Equal, *Turn off Toolbar Help*, *Zoom*, *Application* and *Refresh*

Draw Menu

In the *Draw Mode*, we can draw the geometry on which we want to solve the PDE. As in PDE Toolbox, the IRToolbox provides four types of solid objects: polygons, rectangles, circles, and ellipses by the selecting the following options from the pull-down menu of the **Draw** menu: *Rectangle/Square*, *Rectangle/square (centered)*, *Ellipse/Circle*, *Ellipse/Circle (centered)*, *Polygon*. In order to rotate the selected objects we can use the *Rotate* choice from the same menu.

The objects are used to create a Constructive Solid Geometry model (CSG model). Each solid object is assigned a unique label, and by the use of set algebra, the resulting geometry can be made up of a combination of unions, intersections, and set differences. By default, the resulting CSG model is the union of all solid objects. The only thing that we must take care when we draw the geometry is that the solid objects must not overlap each other if we want to solve the PDE problem using the interface relaxation methodology. The *IRtool* as it is implemented, it deals with geometries that consist of rectangles.

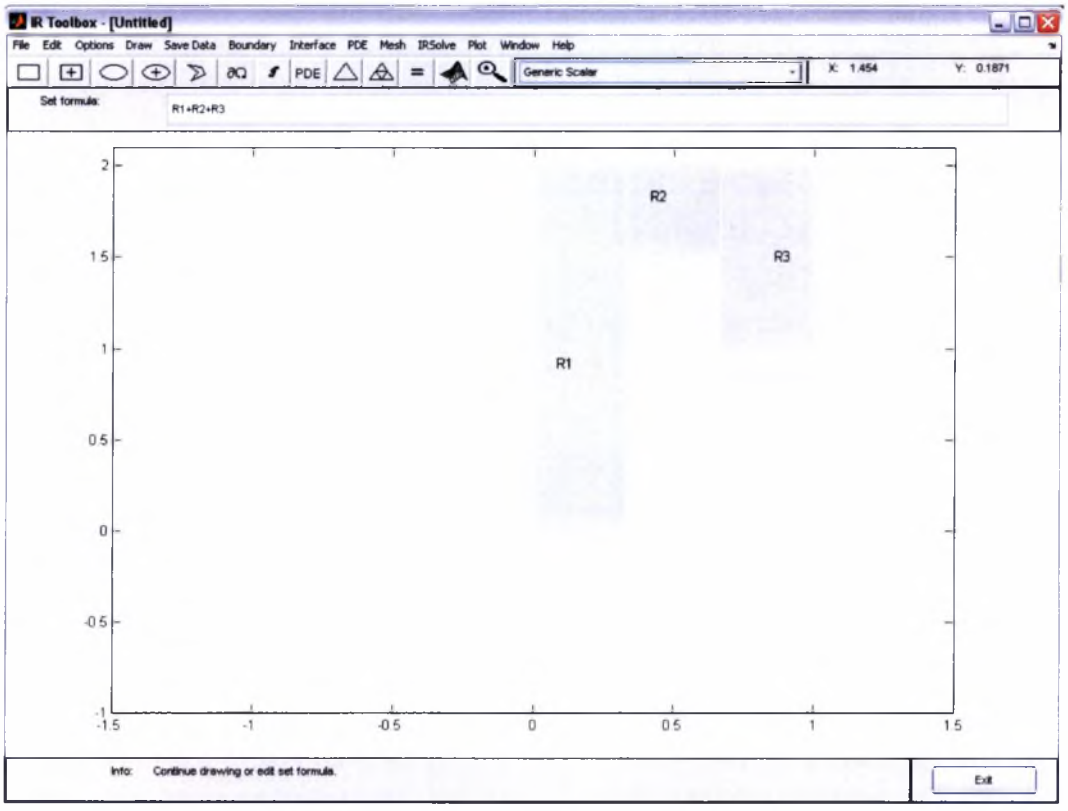


Figure 5. Draw mode.

In the above example (Uniform Case), that we are going to explained it further below (Chapter 7), the resulting CSG model is the union of the three rectangles R1, R2 and R3, described by set algebra as R1+R1+R3 (Figure 5).

If we want, we can save this CSG model as an M-file. We simply use the **Save As. . .** and **Save** option from the **File** menu, and enter a filename of our choice. All the additional steps in the process of modeling and solving your PDE are then saved to the same M-file. This concludes the drawing part.

Boundary Menu

In the **Boundary** menu we have the following options: *Boundary mode*, *Specify Boundary Conditions*, *Show Edge Labels*, *Show Sub-domains Labels*, *Remove Sub-domain Border*, *Remove All Sub-domain Borders* and *Export Decomposed Geometry, Boundary Cond's*.

We can now define the boundary conditions for the outer boundaries. We can enter the Boundary Mode by clicking the $\partial\Omega$ icon, or by selecting **Boundary**

Mode from the **Boundary** menu. We can now remove sub-domain borders and define the boundary conditions. In boundary mode, we did not do any changes and the window is shown below (Figure 6).

The boundaries are indicated by colored lines with arrows. The boundary condition can also be a function of x and y, or simply a constant. By default, the boundary condition is of Dirichlet type: u = 0 on the boundary.

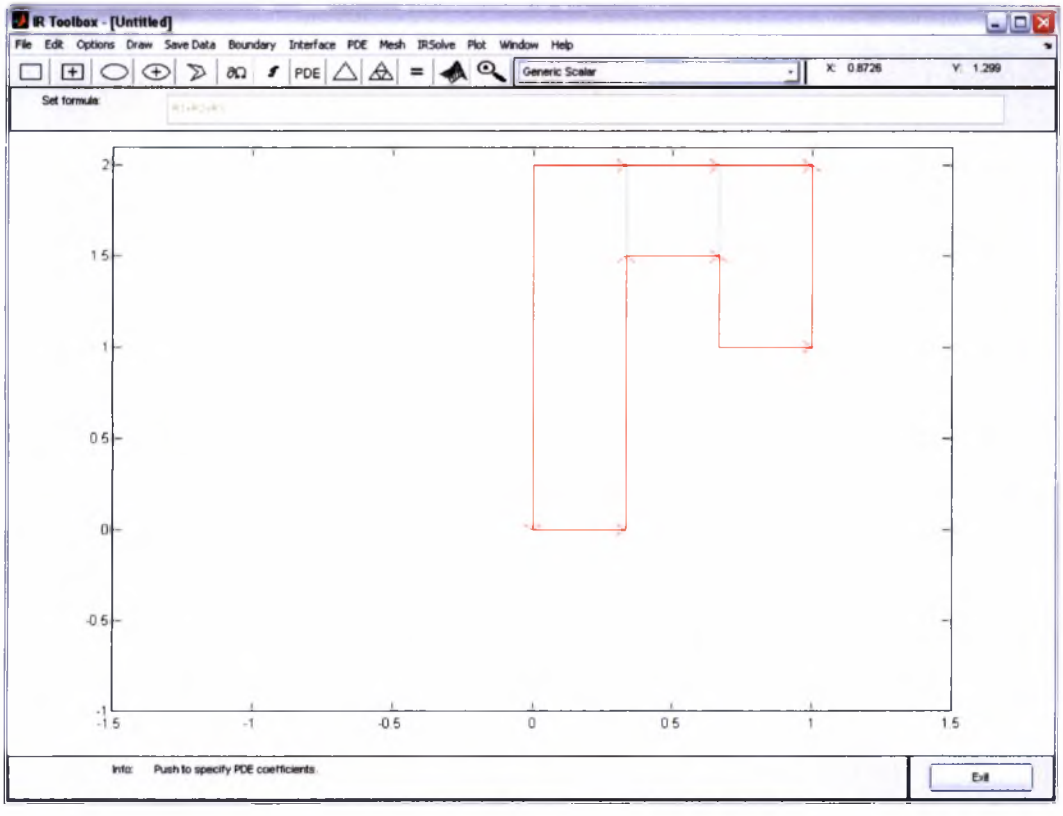


Figure 6. Boundary mode.

Double-clicking anywhere on the selected boundary segments opens the **Boundary Condition** dialog box (Figure 7). Here, we select the type of boundary condition, and enter the boundary condition as a MATLAB expression. The boundary condition is typed in a file named *mytrue.m*. In this file, the user types the elliptic problem that he wants to solve. The elliptic problem has the following form:

$$Lu(x,y) = -\nabla^2 u(x,y) + \gamma^2 u(x,y) = f(x,y),\ (x,y)\epsilon\Omega$$

$$u(x,y) = u^2(x,y),\ (x,y)\ \epsilon\Omega$$

where the right side function $f(x,y)$ and the boundary value function $u^2(x,y)$ are selected if we consider the true solution $u(x,y)$.

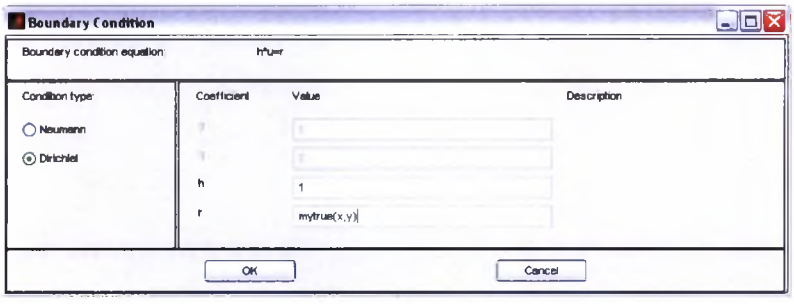


Figure 7. Boundary Condition dialog box.

Interface Menu

The pull-down **Interface** menu is shown below (Figure 8):

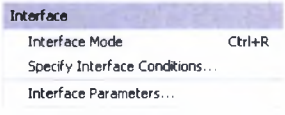


Figure 8. Interface mode: pull-down menu.

Interface mode: Enters the interface mode, where are highlightened only the interface segments.

Specify Interface Conditions: Specify interface conditions for the selected interfaces. If no interfaces are selected, the entered interface condition applies to all interfaces. It displays a dialog box in which, we can specify the interface condition. In the dialog box, we enter the initial values for the interface segment.

Interface Parameters: Displays a dialog box where we can enter the parameters that are needed for the interface condition. For GEO method, the parameter that we should define is ω (omega), and for ROB method we should determine the number λ (lambda).

We can now enter the Interface Mode by clicking the ***I*** icon, or by selecting **Interface Mode** from the **Interface** pull-down menu and we get a window that seems like the Figure 9.

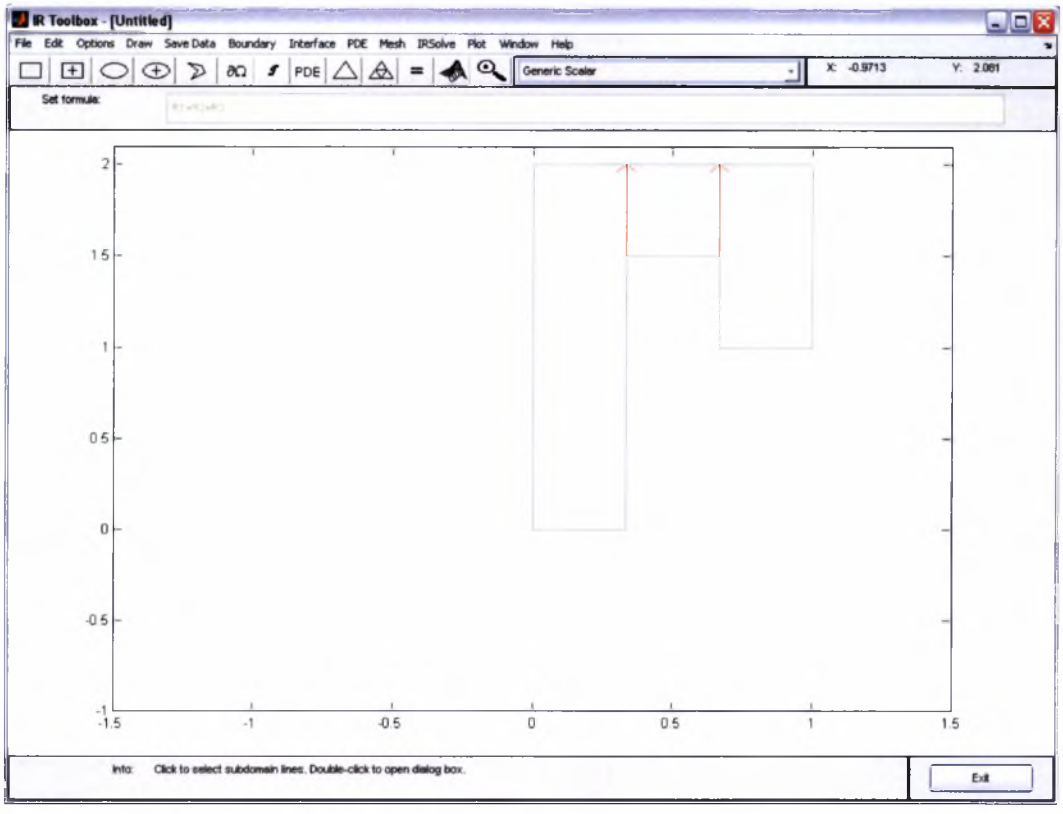


Figure 9. Interface mode.

The **Interface** menu does not exist in PDE Toolbox. We have created it in order to be able to deal with the internal boundary segments that until now were not considered at all in *pdetool*. In order to achieve that, we did some software accessions in the main file (*IRtool.m*). So, when the user enters the interface mode the interface segments are highlighted because Matlab runs the Callback function *IRtool('IRmode')*. This function calls automatically the *IRtool('drawIR')*. Inside *drawIR*, we get the geometry data (assigned as *dl1*) and we choose to turn off everything else and light up only the interface lines by using the command *intbounds=find([dl1(6,:) ∽=0 & dl1(7,:) ∽=0]);*. In the 6^{th} and 7^{th} row of the array dl1 there are the numbers of the right and the left domain. If in any of these rows there is the number 0 (0 assigns the non-domain area), it means that the segment is an external boundary segment.

Double-clicking anywhere on the selected interface segments opens the **Interface Condition** dialog box (Figure 10). Here, we select the type of interface relaxation method, and enter the interface condition as a MATLAB expression.

We can choose between two methods **ROB** and **GEO** and we can change the parameters g and r . For the GEO method, we can define the initial guess of each interface by simply typing the *my*(x,y)* in the box about the r parameter, where * corresponds to the number of the interface that we have selected. The *my*(x,y)* file can easily be modified by the user according to the corresponding problem. The initial guess is the first value that is used in order to compute the true solution on each interface line. The *IRtool* is capable to solve PDE problems that have up to ten interfaces and by using the GEO Interface Relaxation method. However, it is very easy to expand the program for a bigger number of interfaces. And, further research will enable the implementation of ROB Interface Relaxation method.

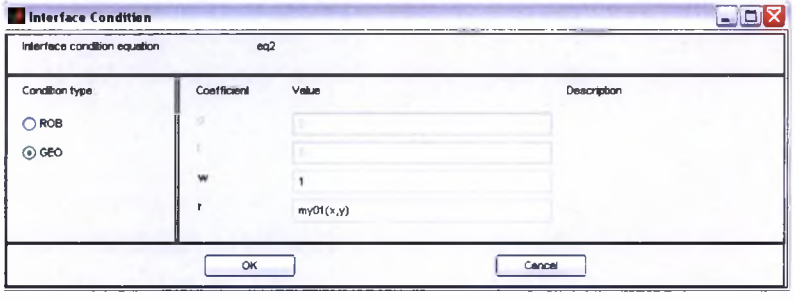


Figure 10. Interface Condition dialog box.

The equation for the interface relaxation method that we have implemented is:

GEO: $u_{L,R}^{new} = \frac{u_L^{old}+u_R^{old}}{2} + \omega(\frac{\partial u_L^{old}}{\partial u} + \frac{\partial u_R^{old}}{\partial u})$

Whenever we choose either ROB or GEO, we change the value in a variable named *IRflag*. This variable is necessary in IRSolve mode (it will be presented later) in order to know which interface relaxation method we will use to solve the PDE problem.

PDE Menu

The **PDE** menu is exactly as it is in Matlab and has the following options: *PDE Mode*, *Show Sub-domain Labels*, *PDE Specification* and *Export PDE Coefficients*.

The elliptic PDE equation is $-\nabla \cdot (c\nabla u) + au = f$. The parameter d does not apply to the elliptic PDE. The coefficients a, c and f can be constants or functions.

The PDE mode for the specific PDE problem that we have is shown in the Figure 11.

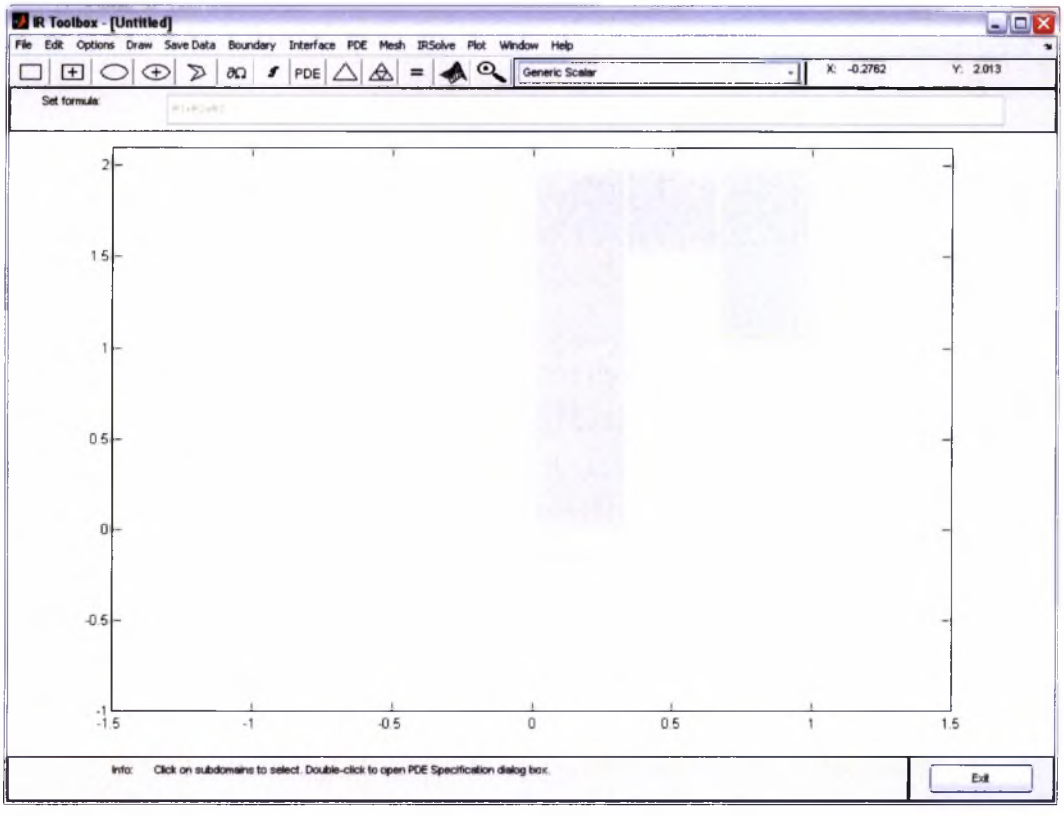


Figure 11. PDE mode.

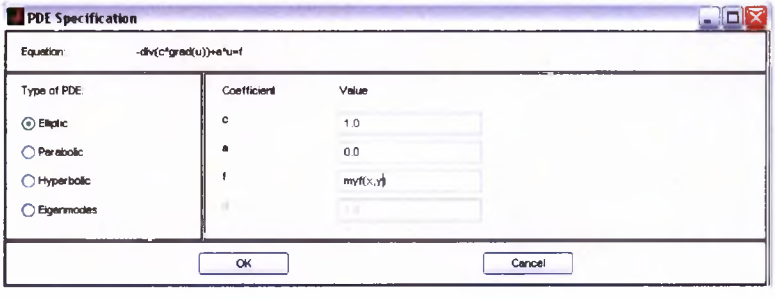


Figure 12. PDE Specification dialog box.

In the dialog box, we can select the type of PDE (elliptic, parabolic, hyperbolic, or eigenmodes) and define the applicable coefficients depending on the PDE type. In the above dialog box (Figure 12), we have an elliptic PDE defined by the following equation:

$$-\nabla \cdot (c \nabla u) + au = f, \text{ with } c = 1.0,\ a = 0.0, \text{ and } f = myf(x,y).$$

The file *myf(x,y)* can easily be modified by the user because it is unique for every differential problem.

Mesh Menu

The **Mesh** menu has the following options: *Mesh Mode*, *Initialize Mesh*, *Refine Mesh*, *Jiggle Mesh*, *Undo Mesh Change*, *Display Triangle Quality*, *Show Node Labels*, *Show Triangles Labels*, *Parameters* and *Export Mesh*.

Parameters for controlling the jiggling of the mesh, the refinement method, and other mesh generation parameters can be found in a dialog box that is opened by selecting **Parameters** from the **Mesh** menu. At this time, we must initialize the maximum edge number, as this refers to the *hmax*. *Hmax* is necessary in order to solve the PDE problem. This menu is the same as it is in PDE Toolbox and the window that appears looks like the Figure 13.

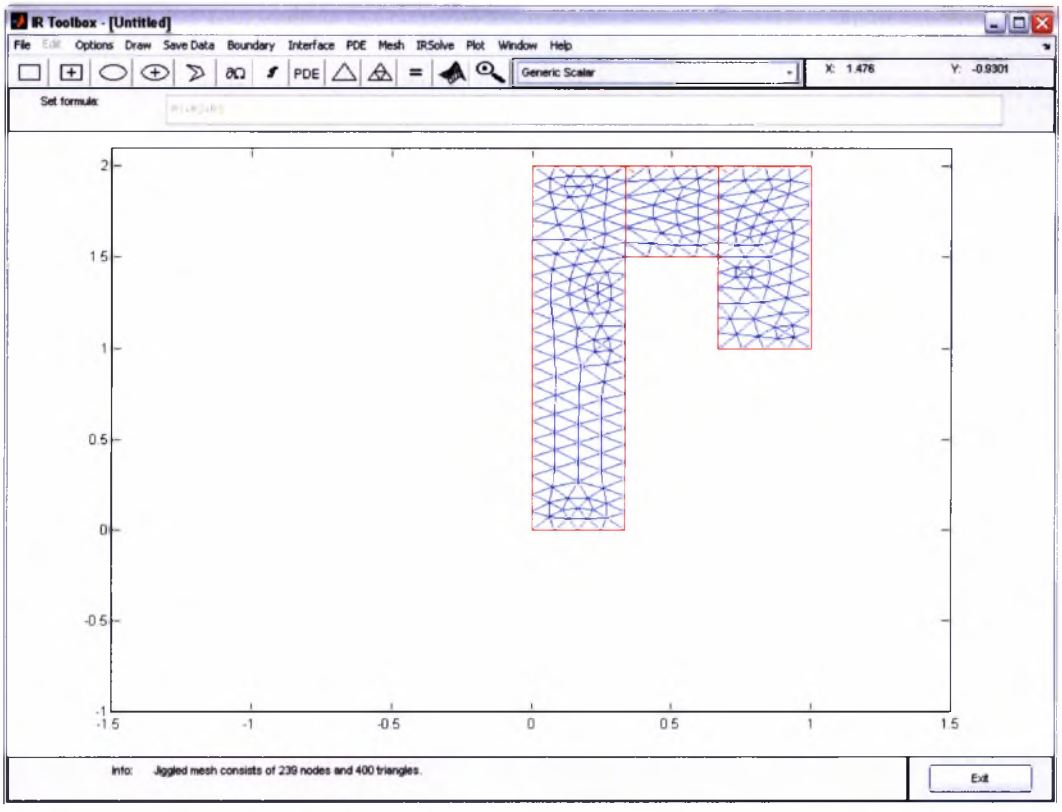


Figure 13. Mesh mode.

Save Data

The **Save Data** menu is a new pull-down menu that we implemented in IRToolbox in order to save some important data in files in case that we need them. The basic idea is to decompose the geometry into domains and divide the information for each domain.

First of all, we decompose the solid objects *GD* (geometry data) into the minimal regions *dl1* by using the function *decsg*. By this way, we have sperate arrays that correspond to different domains. This is very important because in IRToolbox the main concept is to treat the PDE problem as local problems and then combine the local solutions in order to get the global solution for the whole PDE problem. Afterwards, we divide the boundary data (array *bl*), the mesh conditions *(p,e,t)* and the PDE coefficients *(c,a,f,d)*. All the above information is saved into files. Each file is named as *dom*.m*, where * is the number of the domain, and it contains the information about the corresponding domain.

In addition, by using the function *assempde* we get the array u and the derivatives on the nodes *(ux, uy)* for each domain and we save this information in files named *solvedomain*.m*, where * is the number of each domain.

So, by clicking on **SaveData** menu, the user can easily extract all the data that he might need about each individual domain. The files can be found in the same directory folder that he uses at the same time.

IRsolve

The pull-down **IRsolve** menu is shown below Figure 14):

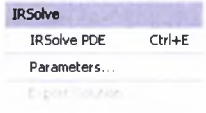


Figure 14. IRsolve mode: pull-down menu.

IRSolve PDE: Solve the partial differential equation for the current CSG model and triangular mesh, and plot the solution and the error.

Parameters: Display a dialog box where we can determinate the basic parameters in order to solve our problem. The parameters that are available are the maximum number of iterations and the tolerance about the error.

Export Solution: Export the PDE solution vector u and, if applicable, the computed eigenvalues l to the main workspace.

We are now ready to solve the problem by clicking the = button or by selecting **IRSolve PDE** from the **IRSolve** menu to solve the PDE. The solution is then plotted. By default, the plot uses interpolated coloring and a linear color map. A colorbar is also provided to map the different shades to the numerical values of the solution.

The **IRSolve** menu differs a lot from the **Solve** pull-down menu of *pdetool*. The basic idea is that we decompose the geometry data *gd*, the boundary data *b*, the mesh conditions *p*, *e*, *t* and the PDE coefficients *c*, *a*, *f*, *d* for each domain that we draw. We save the decomposed objects in cell arrays. Each line of the cell contains the information of each domain. By this way, we can extract every item we need very easily.

Then, we implement the subjective part of IRSolve menu, that it is present by the below algorithm:

```
for i=1, 2, . . . Maximum Iterations
    for j=1, 2, . . .#domains,
        u, ux, uy=solvedomain j
    end
    for l=1, 2, . . . #interfaces
        IRmethod(IRflag,l)
    end
end
```

By solvedomain j, we mean that we call the function *assempde* in order to get the variable *u*. Then we use the functions *pdegrad* and *pdeprtni* and we get the derivatives on the nodes for each different domain. For the loop about the interfaces, we call the function *IRmethod(IRflag,l)*, where *IRflag* is the variable that determinates which method (ROB or GEO) we will use. Afterwards, IRmethod calls the function ROB or GEO for each interface segment. This function returns all the useful information about the interfaces. The main loop continues if we have minimum error or if we reach the maximum number of iterations. By that time we get the true solution of the PDE problem.

The maximum number of iterations and the tolerance can be determined by selecting **Parameters** from the **IRSolve** menu. The window that opens looks like the figure below (Figure 15).

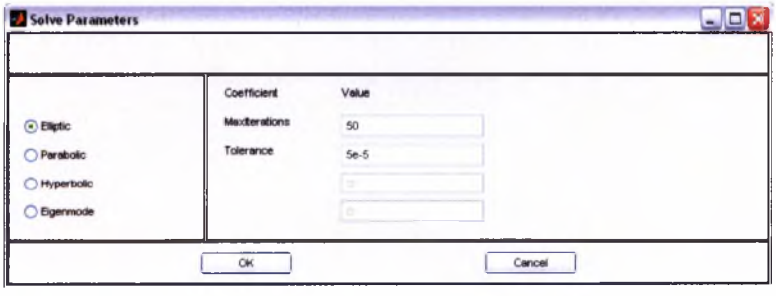


Figure 15. Solve Parameters dialog box.

The solution of the above partial differential equation after 11 iterations is shown in the following Figure 16. In this figure we represent the solution of the PDE problem on the whole geometry data.

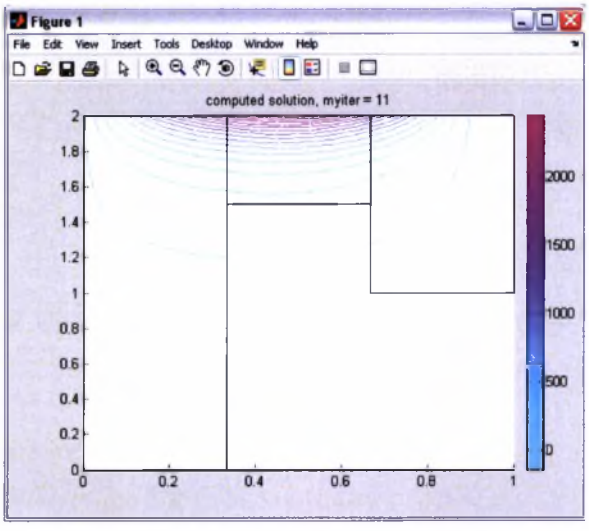


Figure 16. Plot of the solution and the error.

Along with the plot of the solution, the plot of the error function also appears. We can get the error values if we subtract the exact solution of the PDE problem from the computed solution.

In the next figure (Figure 17) we can see the history of the interface values and the error history. For the figure of the history of interface values, with the blue color the true value of the interface is presented, the red line represents the current value of the interface and the black lines corresponds to the past values of the interface. In the figure of the history of the error history, we can see how the error changes as the number of iterations increases.

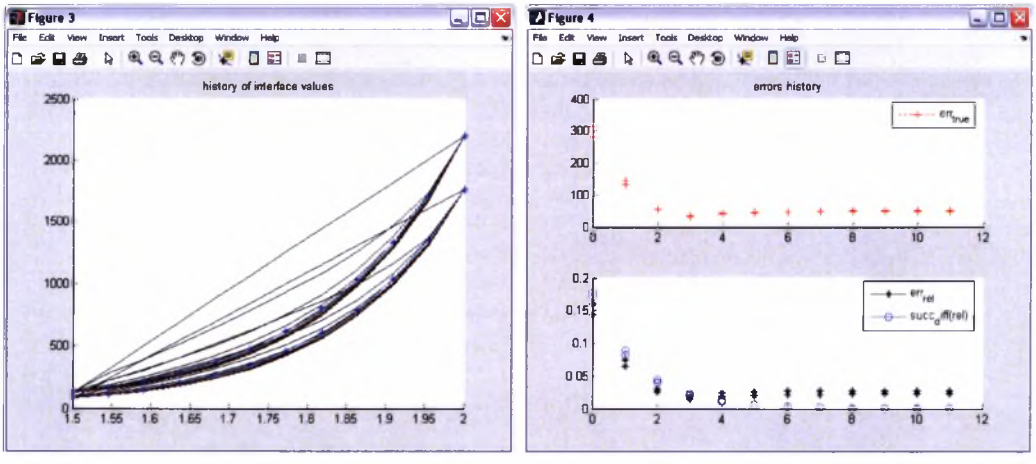


Figure 17. Plot of the interface and the error history.

Chapter 7: Numerical Experiments

In this chapter, three different examples will be presented in order to illustrate the use of the *IRtool*.

Example 1. Uniform Case

In order to understand the Interface Relaxation method, we experiment its applicability and convergence characteristics on geometry decomposition problems. In the first example, we consider the following elliptic problem:

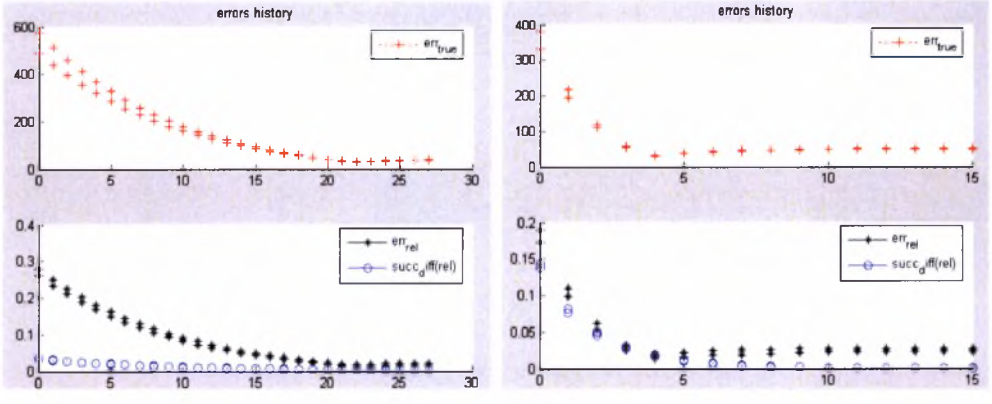
$$Lu(x,y) = -\nabla^2 u(x,y) + \gamma^2 u(x,y) = f(x,y),\ (x,y)\epsilon\Omega$$
$$u(x,y) = u^b(x,y),\ (x,y)\epsilon\partial\Omega$$

The true solution $u(x,y)$ is

$$u(x,y) = e^{y(x+4)}x(x\text{-}1)(x\text{-}0.9)y(y\text{-}0.5).$$

The first PDE problem consists of a geometry that is decomposed into three domains with interfaces on $x_1 = 1/3$ and $x_2 = 2/3$.

First of all, we determined different experiments in order to consider which parameters affect the convergence of the GEO method. The convergence of the method depends only on the relaxation parameter *omega*. This idea is illustrated in the following figures (Figure 18) that show that as we increase the parameter *omega*, the convergence of the GEO Interface Relaxation method is faster.



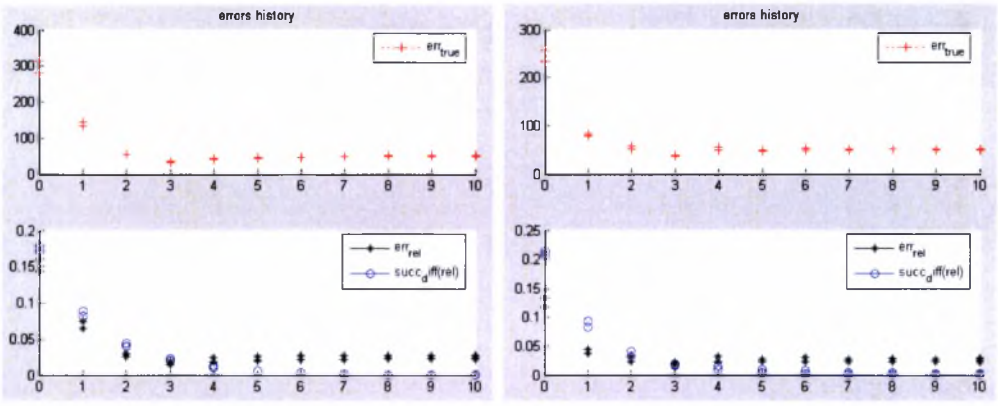
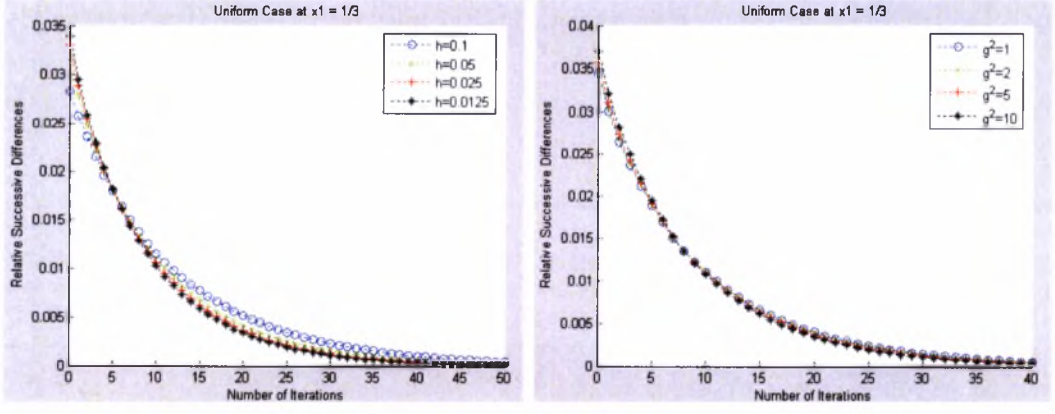


Figure 18. Plots of the error history on the interfaces $x_1 = 1/3$ and $x_2 = 2/3$ for omega $= 0.01, 0.03, 0.05, 0.07$.

We next plot in Figure 19 the convergence history. Specifically, we plot the max norm of the relative difference of two successive iterants on the two interface points as this is depicted by the legends on the left versus the number of iterations. According to these figures, we can easily conclude that the rate of convergence is independent of the local discretization resolution h and depends very little on the PDE coefficient γ^2.



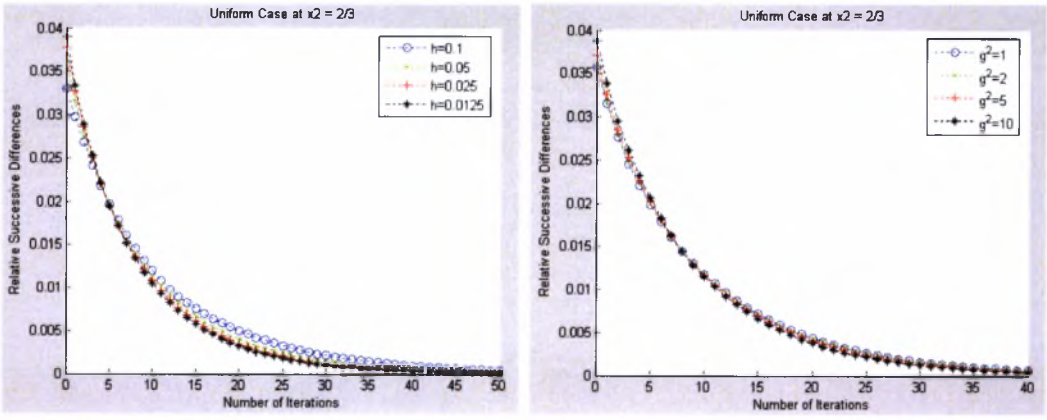


Figure 19. Plots of max norm of the relative difference of two successive iterants measured on the two interfaces of the PDE problem versus the number of iterations. For the plots on the left we set $\gamma^2 = 2$ and $h = 0.1/(2^i)$ for $i = 0, 1, 2, 3$ and for the plots on the right $\gamma^2 = 1, 2, 5, 10$ and $h = 0.05$.

We next plot in Figure 20 the relative true error of the computed solution.

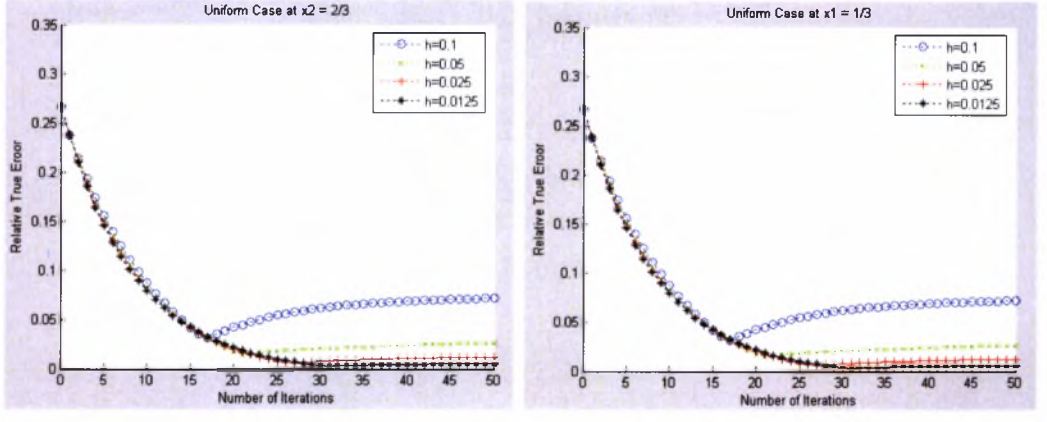


Figure 20. Plots of max norm of the relative true error of the computed solution $||u^{(k)} - u||\infty / ||u||\infty$ versus the iteration number k.

Example 2. Non-Uniform Case

The second PDE problem consists of a geometry that is decomposed into three domains with interfaces on $x_1 = 1/5$ and $x_2 = 1/2$. We use the same differential elliptic equation and true solution $u(x,y)$, as we used in example 1. Below, we have the figures (Figure 21 and Figure 22) for the non-uniform case.

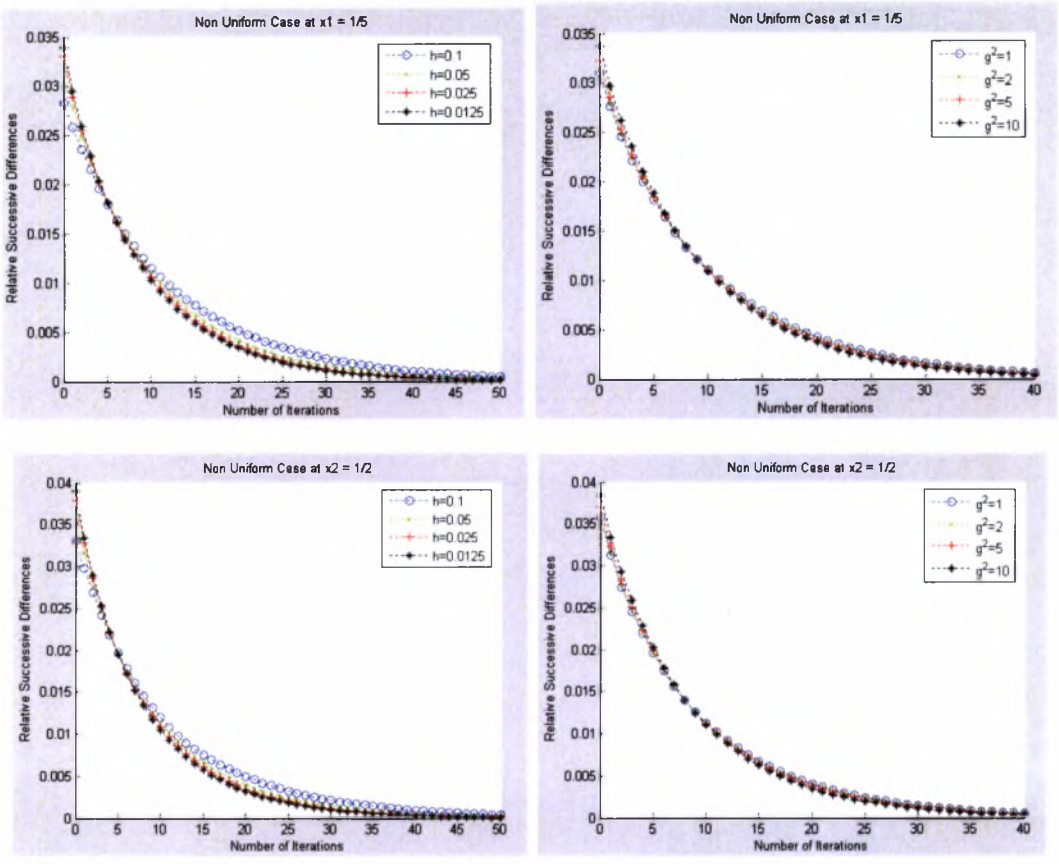


Figure 21. Plots of max norm of the relative difference of two successive iterants measured on the two interfaces of the PDE problem versus the number of iterations. For the plots on the left we set $\gamma^2 = 2$ and $h = 0.1/(2^i)$ for $i = 0$, 1, 2, 3 and for the plots on the right $\gamma^2 = 1$, 2, 5, 10 and $h = 0.05$.

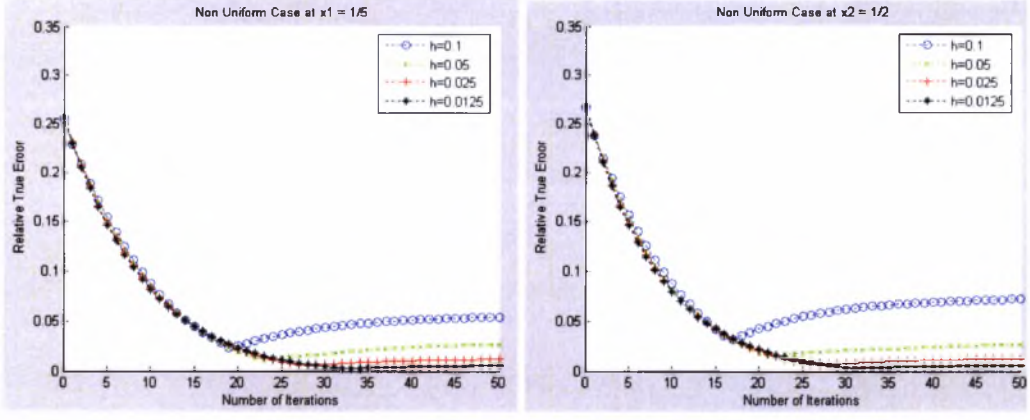
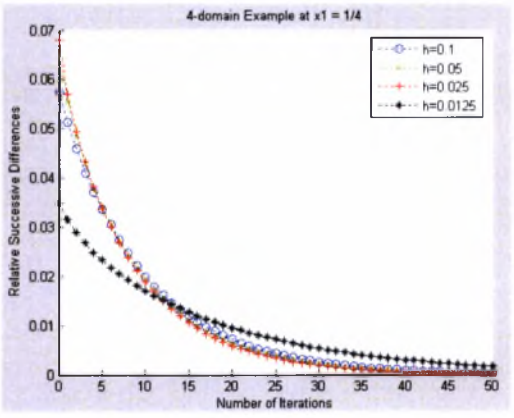
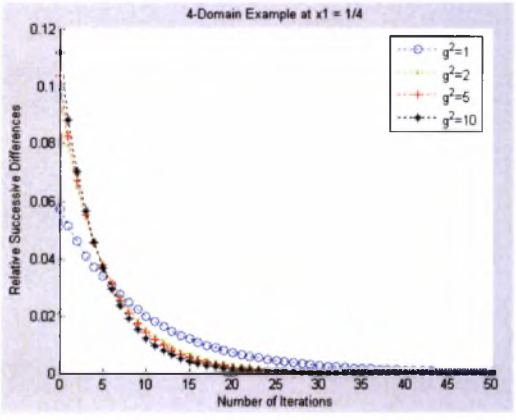


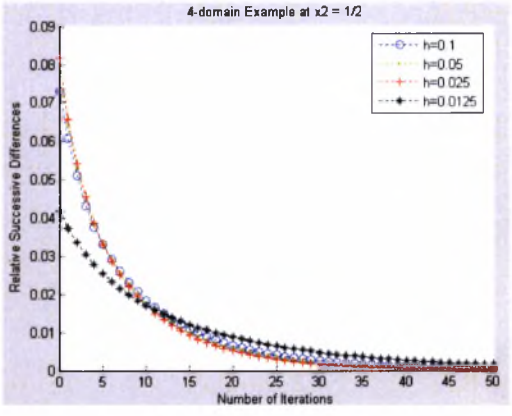
Figure 22. Plots of max norm of the relative true error of the computed solution $||u^{(k)} - u||\propto / ||u||\propto$ versus the iteration number k.

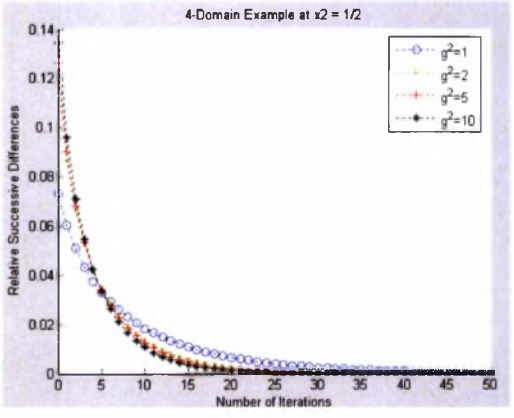
Example 2. 4-Domain Case

The Third PDE problem consists of a geometry that is decomposed into four domains with interfaces on $x_1 = 1/4$, $x_2 = 1/2$ and $x_3 = 3/4$. We use the same differential elliptic equation and true solution $u(x,y)$, as we used in example 1. Below, we have the figures for the 4-Domain example (Figure 23 and Figure 24).









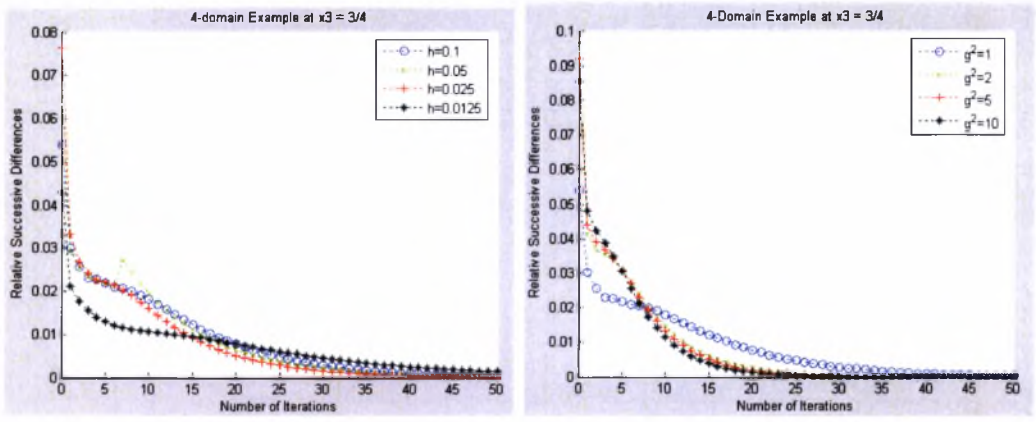


Figure 23. Plots of max norm of the relative difference of two successive iterants measured on the two interfaces of the PDE problem versus the number of iterations. For the plots on the left we set $\gamma^2 = 2$ and $h = 0.1/(2^i)$ for $i = 0, 1, 2, 3$ and for the plots on the right $\gamma^2 = 1, 2, 5, 10$ and $h = 0.05$.

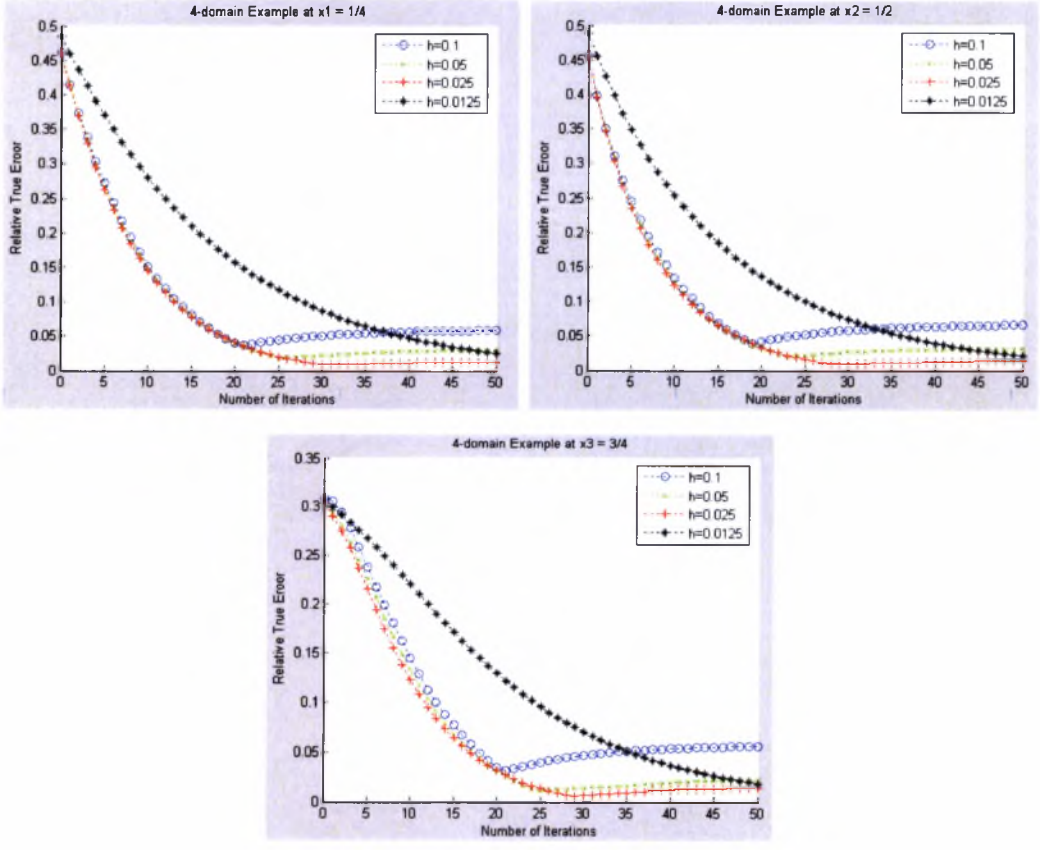


Figure 24. Plots of max norm of the relative true error of the computed solution $||u^{(k)} - u||\infty \; / \; ||u||\infty$ versus the iteration number k.

References

[1] A. Quarteroni, A. Valli. *Domain decomposition methods for partial differential equations.* Oxford University Press, 2000.

[2] Panagiota Tsompanopoulou, *Collaborative PDE Solvers: Theory and Practice.* PhD Thesis, University of Crete, Department of Mathematics, 2000.

[3] P.E. Bjorstad, O. Widlund, *To overlap or not overlap: A note on a domain decomposition method for elliptic problems (Pages: 1053 - 1061).* 1989.

[4] PDE Toolbox, Matlab, http://www.mathworks.com/products/pde.

[5] H.S. McFaddin. *An Object-based Problem Solving Environment for Collaborating PDE Solvers and Editors.*, PhD Thesis, Computer Science Department, Purdue University, 1992.

[6] T.T. Drashansky, *An agent-based approach to building multidisciplinary problem solving environments.* PhD Thesis, Computer Science Department, Purdue University, 1996.

[7] T.T. Drashansky, E.N. Houstis, N. Ramakrishnan, J.R. Rice, *Networked Agents for Scientific Computing.* ACM Communications, 1999.

[8] S.Karin, S.Graham, *The high performance computing continuum.*, ACM Communications, 1998.

[9] T.Finin, Y.Labrou, J.Mayfield, *KQML as an agent communication language.* Cambridge, 1997.

[10] S.McFaddin, J.R. Rice, *RELAX: A software platform for PDE Interface Relaxation methods.*CSD-TR-91-018, 1991.

[11] J.R. Rice, P.Tsompanopoulou, E.Vavalis, *Interface relaxation methods for elliptic differential equations.* Applied Numerical Mathematics 32 (2000) 219-245, Purdue University, Computer Science Department, 2000.

[12] J.R. Rice, P.Tsompanopoulou, E.Vavalis, *Fine Tuning interface relaxation methods for elliptic differential equations.*Technical Report CSD-TR-98-017, Purdue University, Computer Science Department, 2002.